



\$9.95

Display Until
4/15/93

os/2 *Developer*

THE MAGAZINE FOR ADVANCED SOFTWARE DEVELOPMENT

Vol. 5 No. 1



**Spotlight on
Computer
Associates**



**Testing
Your OS/2
Applications**



**OS/2
Application
Migration**



Delivering the Power: WATCOM C9.0/386

► The Widest Range of 32-bit Intel x86 Platforms

32-bit DOS, 32-bit Windows, OS/2 2.0, AutoCAD ADS

► The Industry's Leading Code Optimizer

Advanced global optimizer with new 486 optimizations

► The Most Comprehensive Toolset

Debugger, profiler, protected-mode compiler and linker, 32-bit DOS extender with royalty-free run-time, licensed components from Microsoft SDK, and more

► The Best Value in 32-Bit Tools: \$895*

Unleash 32-bit Power!

WATCOM C9.0/386 lets you exploit the two key 32-bit performance benefits. The 32-bit flat memory model simplifies memory management and lets applications address beyond the 640K limit. Powerful 32-bit instruction processing delivers a significant speed advantage: typically at least a 2x speedup.

You Get:

- **100% ANSI and SAA compatible:** C9.0/386 passes all Plum Hall Validation Suite tests
- **Extensive Microsoft compatibility** simplifies porting of 16-bit code
- **Royalty-free** run-time for 32-bit DOS, Windows and OS/2 apps
- **Comprehensive toolset** includes debugger, linker, profiler and more
- **DOS extender** support for Rational, Phar Lap and Ergo
- **Run-time compatible with WATCOM FORTRAN 77/386**

32-bit DOS support includes the DOS/4GW 32-bit DOS extender by Rational Systems with royalty-free runtime license

- Virtual Memory support up to 32Mb

32-bit Windows support enables development and debugging of true 32-bit GUI applications and DLLs.

- Includes licensed Microsoft SDK components

32-bit OS/2 2.0 support includes development for multiple target environments including OS/2 2.0, 32-bit DOS and 32-bit Windows

- Access to full OS/2 2.0 API including Presentation Manager
- Integrated with IBM Workframe/2 Environment

AutoCAD ADS and **ADI** Development: Everything you need to develop and debug ADS and ADI applications for AutoCAD Release 11

Novell's *Network C* for *NLM's SDK* includes C/386

The Industry's Choice.

Autodesk, *Robert Wenig, Manager, AutoCAD for Windows:*

"At Autodesk, we're using WATCOM C/386 in the development of strategic new products since it gives us a competitive edge through early access to new technologies. We also highly recommend WATCOM C/386 to third party AutoCAD add-on (ADS and ADI) developers."

Fox Software, *David Fulton, President:* "FoxPro 2.0 itself is written in WATCOM C, and takes advantage of its many superior features. Optimizing for either speed or compactness is not uncommon, but to accomplish both was quite remarkable."

GO, *Robert Carr, Vice President of Software:* "After looking at the 32-bit Intel 80x86 tools available in the industry, WATCOM C was the best choice. Key factors in our decision were performance, functionality, reliability and technical support."

IBM, *John Soyering, Director of OS/2 Software Developer Programs:* "IBM and WATCOM are working together closely to integrate these compilers with the OS/2 2.0 Programmer's Workbench."

Lotus, *David Reed, Chief Scientist and Vice President, Pen-Based Applications:* "In new product development we're working with WATCOM C because of superior code optimization, responsive support, and timely delivery of technologies important to us like p-code and support for GO Corp's. PenPoint."

Novell, *Nancy Woodward, V.P. and G.M., Development Products:* "We searched the industry for the best 386 C compiler technology to incorporate with our developer toolkits. Our choice was WATCOM."

WATCOM C/386

32-Bit Optimizing Compiler and Tools

WATCOM C9.0/386

- ▶ 100% ANSI C optimizing compiler
- ▶ Protected-mode compiler ▶ OS/2 hosted-compiler ▶ Royalty-free DOS extender with VMM support ▶ Licensed components of the Microsoft Windows SDK
- ▶ Interactive source-level debugger ▶ Linker
- ▶ Protected-mode linker ▶ OS/2-hosted linker ▶ Profiler
- ▶ Object code librarian ▶ Object code disassembler ▶ MAKE facility ▶ Patch facility ▶ Object module convert utility
- ▶ Windows supervisor ▶ Bind facility for Windows applications
- ▶ 32-bit run-time library object code ▶ Special 32-bit libraries for Windows API ▶ Graphics library for Extended DOS applications ▶ 32-bit Run-time libraries for Windows ▶ 32-bit Run-time libraries for OS/2

Special Offer

Buy **WATCOM C9.0/386** and you'll be eligible to obtain:

WATCOM C9.0 Delta Pack provides you with the tools necessary to develop and debug 16-bit applications for DOS, Windows, and OS/2. **Only \$99.** (\$495 comparative separate cost)

WATCOM FORTRAN 77/386 Delta Pack provides you with everything necessary to develop and debug 32-bit FORTRAN applications for extended DOS, 32-bit Windows and OS/2 2.0. **Only \$399.** (\$895 comparative separate cost)



WATCOM

1-800-265-4555

The Leader in 32-bit Development Tools

415 Phillip Street, Waterloo, Ontario, Canada
 Telephone: (519) 886-3700, Fax: (519) 747-4971
 *Price does not include freight and taxes where applicable. Authorized dealers may sell for less.
 WATCOM C and Lightning Device are trademarks of WATCOM Systems Inc.
 DOS/4G and DOS/16M are trademarks of Rational Systems Inc.
 Other trademarks are the properties of their respective owners.
 Copyright 1992 WATCOM Products Inc.

Keeping your User Interface Options Open with Dialog System from Micro Focus.



Developing the right user interface for your business application is crucial, but how do you know which 'industry-standard' will offer you the most flexibility? Is it Microsoft® Windows™? OS/2® Presentation Manager™? Or perhaps just Character Mode interfaces for DOS and UNIX®?

It doesn't matter...because with Dialog System™ from Micro Focus, you can develop GUIs that will run unchanged in all of those environments, adapting to the look and feel of your target.

Dialog System supports high-level development of graphical and character-based user interfaces. It isolates screen and keyboard logic from the COBOL application's code, replacing hundreds of lines of screen definition with a simple CALL statement. Which, in turn, means smaller, and faster GUI applications that can be easily maintained.



Dialog System also keeps your options open. Graphical user interfaces can be prototyped, developed, even customized without impacting the program code...and those same interfaces will run unchanged even in emulation mode in character-based environments.

Combine flexibility and performance and the solution to the graphical user interface development puzzle is simple: Dialog System.

Keep your user interface options open with Dialog System. Call 800-872-6265 and discover "A Better Way of Programming™" with Micro Focus.

MICRO FOCUS®

Micro Focus Inc., 2465 East Bayshore Road, Palo Alto, CA 94303. Tel: (415) 856 4161.

Micro Focus is a registered trademark and Dialog System and "A Better Way of Programming" are trademarks of Micro Focus Inc. All other products are trademarks of their respective companies.
GSA Contract Number GS00K90AGS5251-PS02.

IBM OS/2 Developer

THE MAGAZINE FOR ADVANCED SOFTWARE DEVELOPMENT



WINTER 1993



EDITOR'S COMMENTS

Delivering More Beef



SPOTLIGHT

Computer Associates' Success: Platform-Adaptive Applications



32-BIT

Designing An OS/2 Device Driver



LAN

LAN Interoperability with OS/2 2.0



TOOLS

Softool: Software Change Management on the PC

AlertVIEW: Network Management

PMfax: Device-Independent Fax Services for OS/2 2.0

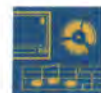
Why PL/I



CLIENT/SERVER

Testing Client/Server Applications

Testing in the GUI and Client/Server World



MULTIMEDIA

OS/2 2.0 Video Systems: Structure and Behavior



INTERNATIONAL

OS/2: The International Scene



GUI

Programming the OS/2 Container Control: The Basics

Implementing a Wastebasket in the Workplace Shell

Thinking in Presentation Manager

Demystifying Custom Controls



DATABASE MANAGER

Developing Applications with Query Manager

5

8

16

25

39

75

80

85

43

53

63

69

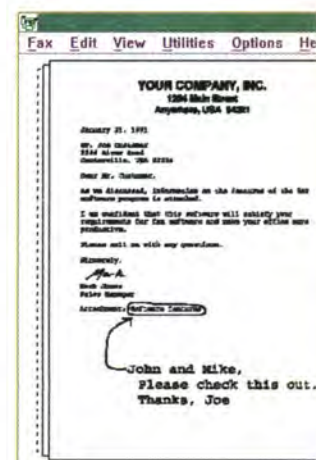
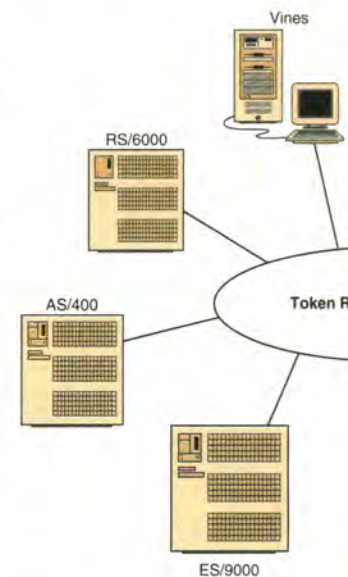
96

102

112

120

132



IBM OS/2 Developer

THE MAGAZINE FOR ADVANCED SOFTWARE DEVELOPMENT

EDITOR

Dick Conklin

EXECUTIVE EDITOR

Nicole Freeman

PRODUCTION EDITOR

Peter Altenberg

COPY/PRODUCTION EDITOR

Lisa Gluskin

GROUP DIRECTOR

Don Pazour

NATIONAL SALES MANAGER/ ADVERTISING SALES STAFF

Veronica Smith

Jo Ben-Atar

Michele Blake

(East: 212-683-9294)

REGIONAL SALES MANAGER/ ADVERTISING SALES STAFF

Cathy Passage

Pam D. Grossman

Michele Anet

(West: 415-905-2392)

MARKETING MANAGER

Susan McDonald

ART DIRECTOR/MARKETING

Christopher H. Clarke

ADVERTISING PRODUCTION COORDINATOR

Kathy Bruin

DIRECTOR OF PRODUCTION

Andy Mickus

DIRECTOR OF CIRCULATION

Jerry Okabe

CIRCULATION MANAGER

Kathy Henry

SUBSCRIPTION INQUIRIES

U.S.: (800) WANT-OS2

Foreign: (708) 647-5960

IBM OS/2 Developer is published by the Personal Systems Line of Business of International Business Machines Corp., Boca Raton, Fla., U.S.A., Dick Conklin, Editor. The terms "OS/2" and "OS/2 2.0" are used in this publication as abbreviations for the full term "OS/2 operating system." OS/2 is a registered trademark of the IBM Corporation.

Titles and abstracts, but no other portions, of information in this publication may be copied and distributed by computer-based and other information service systems. Permission to republish information from this publication in any other publication of computer-based information systems must be obtained from the Editor.

IBM believes the statements contained herein are accurate as of the date of publication of this document. However, IBM, hereby disclaims all warranted either expressed or implied, including without limitation any implied warranty of merchantability or fitness for a particular purpose. In no event will IBM be liable to you for any damages, including any lost profits, lost savings or other incidental or consequential damage arising out of the use or inability to use any information provided through this publication even if IBM has been advised of the possibility of such damages, or for any claim by any other party.

Some states do not allow the limitation or exclusion of liability for incidental or consequential damages so the above limitation or exclusion may not apply to you.

This publication may contain technical inaccuracies or typographical errors. Also, illustrations contained here may show prototype equipment. Your system configuration may differ slightly.

This publication may contain articles by non-IBM authors. These articles represent the views of their authors. IBM does not endorse any non-IBM products that may be mentioned. Questions should be directed to the authors.

This information is not intended to be an assertion of future action. IBM expressly reserves the right to change or withdraw current products that may or may not have the same characteristics or codes listed in this publication. Should IBM modify its products in a way that may affect the information contained in any user of the modification. It is possible, that this material may contain reference to, or information about, IBM products (machines and programs), programming or services that are not be construed to mean that IBM intends to announce such products, programming or services in your country.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. All specifications are subjects to change without notice.

With respect to advertising material herein, this publication does not constitute an expressed or implied recommendation or endorsement by IBM of any particular product, service, company, or technology.

IBM takes no responsibility whatsoever with regard to the selection, performance or use of the products advertised herein. All understanding, agreements, or warranties must take place directly between the software vendors and prospective users.

To correspond with the IBM OS/2 Developer, please write to the Editor at IBM Corp., Internal Zip 2230, P.O. Box 1328, Boca Raton, FL 33429-1328.

IBM, OS/2 2.0, PS/2, Micro Channel, C/2, Presentation Manager, CUA, SAA, Workplace Shell, WIN-OS/2, AS/400, ES/390, NetView, SAA, AD/Cycle, C Set/2, 386, 486, PL/I Package/2, WorkSet/2 are registered trademarks of IBM Corp.

Windows, MS-DOS, and Code View are registered trademarks of Microsoft Corp.

CA-SORT, Virtual Machine, IDOS/VS, EXECMOD, SYMBUG, CA-SORT, Virtual Machine, IDOS/VS, EXECMOD, SYMBUG, CA-Textor, CA-Compete, CA-dBfast, CA-SuperProject, CA-REALIZER, and CA-UNICENTER are trademarks of Computer Associates Inc.

CompuServe is a registered trademark of CompuServe Inc.

UNIX is a registered trademark of AT&T Bell Laboratories.

Network Application Support is a registered trademark of Digital Equipment Corp.

Macintosh is a registered trademark of Apple Computer Inc.

Lotus 1-2-3 is a registered trademark of Lotus Development Corp.

Netware is a registered trademark of Novell Corp.

Banyan and VINES are registered trademarks of Banyan Systems Inc.

Softool, CCC, CCC/Manager, and CCC/Bridge are trademarks of Softool Corp.

SQA:Robot and SQA:Manager are registered trademarks of Software Quality Automation.

Softbridge Automated Test Facility is a registered trademark of Softbridge Inc.

AlertView is a registered trademark of Shany Inc.

SIGMA and SIGMS Engine are registered trademarks of ABB Infosystems.

PMfax is a trademark of Keller Group Inc.

SatisFAXtion is a registered trademark of Intel Corp.

Open+Build, Open+Editor, and DL Language are trademarks of Open+Voice Inc.

IBM OS/2 Developer: Vol. 5, No. 1, Jan. 1993

Subscription information: U.S.: One year (four issues), \$39.95. Canada, Mexico, and international surface mail, add \$16 per year for postage. Canadian GST no. 124513185. International air mail, add \$30 per year for postage. Foreign orders must be accompanied by payments in U.S. funds. For new orders and customer service, call (800) WANT-OS2 (800-926-8672) or (708) 647-5960 (fax: 708-647-0537). IBM employees and branch office customers can subscribe to the IBM OS/2 Developer through IBM Mechanicsburg's Systems Library Services (SLSS) using the IBM OS/2 Developer's order number: G362-0001. **POSTMASTER:** Please send address changes to IBM OS/2 Developer, P.O. Box 1079, Skokie, IL 60076-8079. Allow eight weeks for subscriptions to begin. The IBM OS/2 Developer is published quarterly by IBM Software Support, Internal Zip 2230, 1000 N.W. 51st St., Boca Raton, Fla. 33429, (407) 982-6408, fax (407) 443-4233. Application to mail at second class postage rates is pending at San Francisco, CA, and additional mailing offices.

© Copyright 1992 by International Business Machines Corp. Printed in U.S.A.

Editor's Comments



As promised, we started an OS/2 Developer topic on CompuServe™ (look for us in the OS2DF2 forum, section 13). Thanks to everyone for your praise, encouragement, and ideas. A common thread in many notes was your desire for more "beef" in the articles. We don't have space to print all of our mail, but here are some excerpts:

Iwould hope that future issues would have more articles on the "picky details" of programming for OS/2™ along with code. There are a great number of developers out here who are totally new to OS/2, as well as a number of semi-experienced developers. Space has to be devoted to both groups. You should also cover CSet/2. Further, OS/2 Developer should deal with real-world problems. Developers are hungry for useful information, and you have to fill that need. By the way, quarterly is far too long between issues. At the very least, they should be bimonthly. Programmers need information now—tomorrow may be too late! —*Tony Costanza*

More "beef" ... yes yes yes!...
—*Sheila S. Ruth*

Iwould like to see OS/2 Developer have more technical articles of some depth, accompanied by a lot of source code (with code available in electronic form as well).

The market for in-depth technical information about OS/2 programming is scarcely saturated. Containers. Notebooks. Programming the COM port. Printing to the spooler. Managing fonts. Multi-threading techniques and semaphores to manage threads. Intro to WPS programming. Fast screen redraw with GpiBitBlt. Converting from 16-bit to 32-bit OS/2. Converting from DOS to OS/2. Converting from Windows™ to OS/2

PM. Converting from C to C++. Using pipes and queues. Using DDE. Using various data formats for the clipboard. Mixing text colors and fonts in a single document. Design issues and CUA '91. Optimizing PM programs. Window management (sizing, placement, Z-ordering, owner and parent relationships, HWND_OBJECT use, and so on). Using the new font and file dialogues. Sliders. Value sets. Device drivers. Debugging multi-threaded programs. Debugging PM programs. Creating and using DLLs.

All of these are how-to articles. I would also like survey articles of available programming tools, including GUI builders, C++ class libraries, CASE tools, editors, and so on. These should be hands on pieces, not just a regurgitation of features—how are the tools to use? —*Wayne Kovsky*

Ilike the magazine quite well; I just wish there was more of it. I don't know if monthly is the answer, but six issues a year would be an improvement. I like the articles by ISVs; I don't mind the hype too much, since most of them have been revealing as to what the development staff went through. Others provide insight into ways OS/2 is being used in the real world. Keep up the good work.
—*Dave Bennett*

I'd like to see more PM, WPS, and SOM source code. You've started off pretty well with the SOM stuff. It would be great to have something like a Orfali and Harkey-type column that progresses from a simple ClubMed type of system and eventually gets more complex from issue to issue.

I keep my source code magazines forever. Applicable source code from cover to cover would make a large stack of OS/2 Developers



Dick Conklin

You asked for more "beef," and we're going to deliver it.



part of the Fishman clutter. And my subscription never lapses on that type of magazine! —*Rick Fishman*

First, let me compliment you on the excellent job you have done on *OS/2 Developer*. It has always been most impressive and helpful in providing high-level articles.

However, the past focus has been on high- and medium-level discussions of features as well as ISV articles. I don't think either of these quite constitute the "beef" that has been asked for. What developers want to see is, I believe, working code and a detailed discussion of how to write programs. I would like to see complete code for all articles uploaded to CompuServe as complete, working, sample programs. Printing only key abstracts in the magazine is fine, but a more complete example would be most helpful. —*Guy Scharf*

Good to have you up here. I strongly agree with all the "beef" requests. How about a second edition of *OS/2 Notebook*? —*Jon Wright*

Thanks, everyone! You asked for more beef, and we're going to deliver it. We're also asking our writers to upload the source code of their sample programs on CompuServe. We're almost finished with the second edition of the OS/2 Notebook, a compilation of articles from this magazine, due in early 1993. We are also planning further changes to the magazine, including a new format designed to deliver more articles in each issue and a possible increase in publication frequency.

If you haven't joined us on CompuServe, please drop by and say hello. We'd like to hear from you.

Dick Coll



No Matter What Your GUI Destination **CASEWORKS** Can Take You There



Now You Don't Have to Worry About Choosing the Wrong GUI Development Path...CASEWORKS Lets You Easily Change Direction — by Switching Languages or Platforms — Without Wasting Any Time!

All of the choices surrounding Graphical User Interfaces (GUIs) can make picking a GUI development path seem impossible. Deciding between Windows™ or Presentation Manager™ (PM) is only the first step toward building client/server applications. You need a tool that lets you generate code to Windows or PM for a variety of languages and APIs, and that lets you migrate interfaces among languages and platforms if you change your mind. *CASE:W™ and CASE:PM™ are the only tools that give you that flexibility.*

CASE:W for Windows Development

Windows developers face the challenge of migrating from C to C++ and class library programming. CASE:W makes it easy by generating source code for the three "standards" of Windows programming: the Windows C API, Microsoft Foundation Classes™ and Borland's ObjectWindows™.

Simply buy CASE:W along with a "Knowledgebase" for C, or C++ and a class library. Use the visual designer to create and test your interface. Then generate expert-level, tightly-written, thoroughly-documented code from your knowledgebase. To use the interface in a different environment, simply buy that knowledgebase and generate again. You can even migrate the interface to PM.

CASE:PM for Presentation Manager Development

When you're developing "mission-critical" applications for OS/2™ 2.0, you need a strategic tool that can build PM applications for both 16- and 32-bit PM environments. Our new CASE:PM VIP tool gives you a head start on building client/server applications, and lets you easily migrate legacy applications to OS/2 2.0. Plus, you can leverage all the advanced capabilities of CUA '91.

The Safest Development Route

With CASEWORKS, you get the security of using the leading GUI development tools. Microsoft® includes a version of CASE:W with its QuickC™ for Windows, and more corporations use CASE:PM for developing strategic PM and client/server applications than any other standard language tool. Both tools generate code without restrictions, letting you add any code, anywhere. Plus, a regeneration facility preserves your changes. And you control the generated code, with no proprietary languages, runtimes or licensing fees.

Don't avoid making a decision because you're uncertain which way to go. Choose the only company that can take you to Windows or PM, through any development route: CASEWORKS.

CASEWORKS™

1 Dunwoody Park, Suite 130, Atlanta, GA 30338
1-800-635-1577 • (404) 399-6236 • Fax (404) 399-9516



Spotlight on Computer Associates

CA's Success: Platform-Adaptive Applications



Eric Nelson

by Eric Nelson

Computer Associates International Inc. (CA) is a leading provider of integrated systems management, database management, application development, and business applications software. CA, which creates programs that operate on mainframe, midrange, and desktop computers, is one of the few companies in the world to have such a broad coverage of applications for every major computing platform.

In 1976, Charles B. Wang, Russ Artzt, and Judith Cedenio founded Computer Associates in New York City. Wang is currently CA's chair, Artzt is executive vice president for research and development, and Cedenio is the company's corporate services director. CA moved its headquarters to Islandia, New York, in early 1992.

Mainframe	Midrange	Desktop
IBM MVS, VSE, VM	Digital VAX/VMS	OS/2
Digital VAX/VMS	OS/400	DOS
Fujitsu MSP	HP/UX	Windows
Tandem GUARDIAN 90	Sequent	Macintosh
	SUN Solaris	UNIX
	Other UNIX	

Figure 1: Platforms covered by CA products

CA's first product was CA-SORT™, a sort utility for IBM's Virtual Machine™ (VM) mainframes. By the end of its first year, the

company had over 200 clients and was marketing an assortment of VM utilities, such as IDOS/VS™, EXECMOD™, and SYMBUG™. Over the next few years, CA created or acquired more products and migrated them to other mainframe environments and midrange computers. The company then entered the desktop market, creating applications for UNIX™ workstations, DOS- and Windows™-based personal computers, and the Macintosh™. Currently, Computer Associates offers well over 300 products covering every major computing platform. Platforms supported are listed in Figure 1. Most of its products are compliant with the 1991 Systems Application Architecture™ (SAA) Common User Access™ (CUA) guidelines.



CA founder Charles Wang

Today, with over 90% of Fortune 500 companies using CA software, CA's revenues for 1992 exceeded \$1.5 billion. The company has 7,400 employees and operates in 27 countries, with over 40% of its sales coming from overseas. Distribution channels include retail stores, mail order, value-added resellers, and a corporate sales force.

OS/2 PRODUCTS

Computer Associates is porting seven products to OS/2 2.0, six from Windows, and one from UNIX:

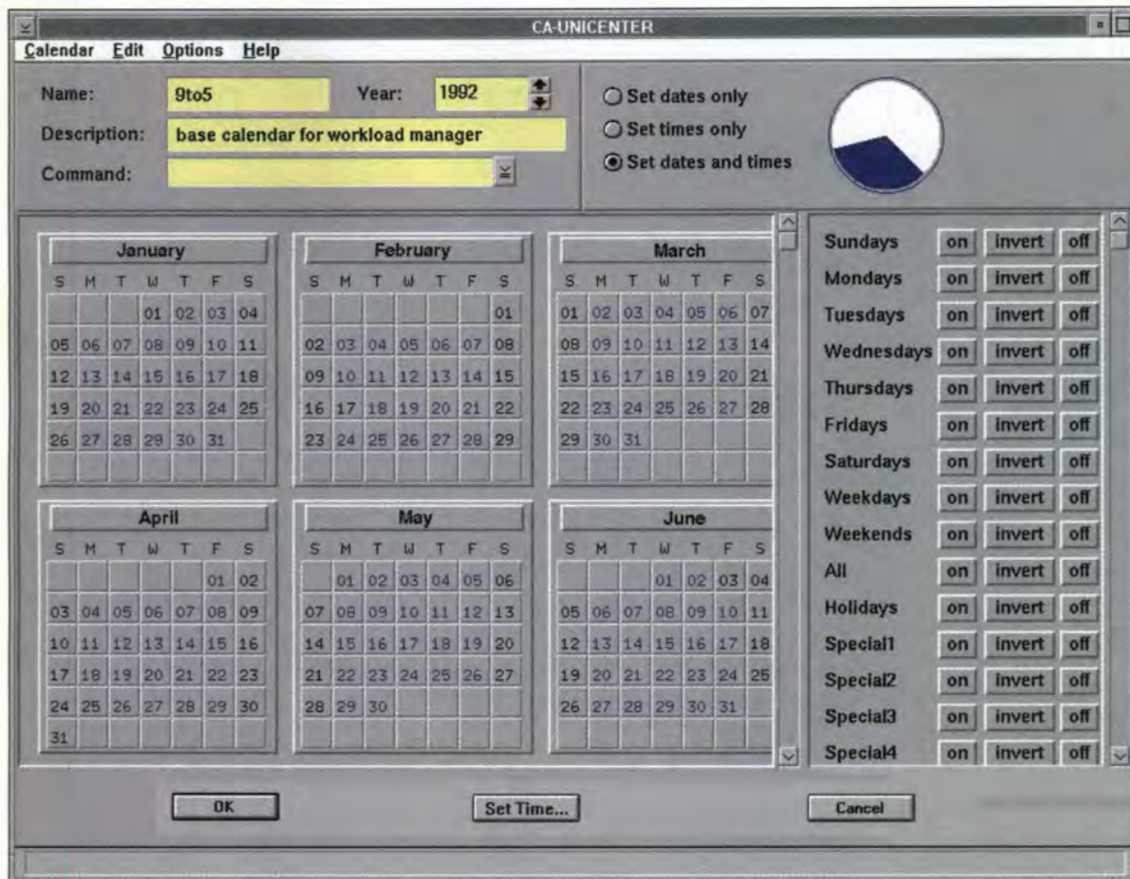


Figure 2: CA-UNICENTER user interface

- ACCPAC Simply Accounting™, an accounting package for small businesses
- CA-Textor™, the company's word processor
- CA-Compete!™, a data modeling and decision support tool that combines a database and a multidimensional spreadsheet
- CA-dBFast™, a graphical Xbase language and development tool
- CA-SuperProject™, CA's project management tool
- CA-REALIZER™, a graphical BASIC development tool
- CA-UNICENTER™, a comprehensive system management tool that includes security, storage management, production control, scheduling, report distribution, resource accounting, performance monitoring, data center automation, and LAN management for NetWare and LAN Services. CA-

UNICENTER, the company's most popular product, has until now been available only on mainframe and midrange computers. A screen from CA-UNICENTER for OS/2 is shown in Figure 2; user interfaces such as this are similar across platforms.

COMPUTER ARCHITECTURE FOR THE 90S

Computer Associates' programming philosophy is "to provide consistent, integrated functionality across products and across platforms with a platform-adaptive user interface." One of the keys to its cross-product and cross-platform consistency, and to CA's ability to port rapidly, is the company's central design philosophy: "Computer Architecture for the 90s," or CA90s.

CA90s is a software architecture designed to:

- Keep as much code as possible platform-independent.

CA's central design philosophy: a "computer architecture for the 90s."



- Put as much code as possible in common service layers available to all applications.
- Provide for communications between, and integration of, applications running on the same or different platforms.

CA90s is a methodology for "platform-adaptive" behavior—rather than simply porting



Figure 3:
User Interface
and Visualization Services Layer

CA90s encourages developers to reuse code by insulating the parts of the application that change from platform to platform.

Windows applications to OS/2 2.0, CA enhances them to best use special features of the target platform. When CA programs are ported to OS/2 2.0, for example, enhancements include multithreading, taking advantage of the Workplace Shell™, and using the System Object Model (SOM) object-oriented programming model. Marc Sokol, director of product strategy for CA, explains, "The most positive response we've heard to the announcement of our OS/2 product line is that we weren't just taking Windows programs and having them operate like Windows programs under OS/2. We are exploiting some of the features of OS/2."

CA90s gives CA the capability to migrate quickly from platform to platform and speeds the time to market by minimizing and isolating code that must be changed. Porting parts of the code to the new platform without changes saves time over the entire development cycle, reducing the time spent on quality assurance, testing, and so on.

CA90s encourages developers to reuse code by insulating the parts of the application that change from platform to platform. "To best reuse pieces of software, before you write the software, think about how the pieces are going

to connect," advises Sokol. "For example, some of the obvious things are separating operating system dependencies or database access from application code. When CA reengineered its Windows products for OS/2, much of the code—even code with heavy user interface content—was ported without changes."

Says Sokol, "One of the first things we do, even with products we acquire, is to ask two questions: how do we immediately start integrating CA90s services into this product, and are there components of the product itself that could become a CA90s service? For example, CA-Textor: on one side, we are looking to make sure that

the user interface and data access portions are separated from the application code. On the other side, we recognize that there's a real value to having a Rich Text Format editing engine. So what we're doing with CA-Textor, at the same time a lot of the other development is going on, is turning CA-Textor into a dynamic link library that can be reused by other components. So if CA-REALIZER or another development tool needs a rich text format editing engine, it can use it from CA-Textor. It's a two way situation... as we acquire technology, we look at it to see how it best fits into those layers and how it can provide services we don't already have."

CA90S SOFTWARE ARCHITECTURE

The CA90s architecture divides code into application code and its Common Services code. Common Services, available to all CA applications, can be thought of as layers of code that lie between the application and the hardware. As much code as possible is put into the Common Services layers, and all platform-dependent code is isolated.

There are four layers that make up the CA90s architecture, as shown in Figure 3:

- User Interface And Visualization Services

- Enterprise Software Solutions
- Integration Services
- Distributed Processing Services.

Enterprise Software Solutions are CA's applications—systems management, business applications, and application development tools. The other three items are common services layers.

User Interface and Visualization Services

This layer contains the code for communication between the user and machine. With the user-interface code located in a discrete layer, an application can run virtually unchanged in even the most advanced graphical environment. The user-interface layer also guarantees a common look and feel across the CA product line. Conformance to a user interface standard such as CUA is coded only once, in this layer; applications can access the code as needed.

The user interface and visualization services layer (shown in Figure 3) supports all major industry standards including CUA, Digital Equipment's Network Application Support™ (NAS), Apple's Macintosh, and the various UNIX standards. The technology, previously used only for CA internal development, will be made available in the form of class libraries as CA adopts object orientation for most products. The layer consists of four components:



Figure 4:
Integration
Services layer

- User Interface Management Services, the foundation of the layer, provides a consistent look and feel across CA products and platforms.
- Graphics provides CA applications with graphics service through a single graphics interface.
- Reporting provides a powerful and flexible report-generating engine.
- Voice allows the use of voice technology.

Integration Services

This layer, shown in Figure 4, supports integration among CA's software products, increasing coordination between applications. It consists of five components:

- Application Services provide a range of high-level services such as project management, change control, and expert systems.
- Event Notification replaces the multiple operating systems that CA applications run on with a single, shared interface to the operating system. Event Notification monitors system and application events and shares this information among applications.
- Security controls and monitors access to all system and application resources and processes.
- Database Management Services provide a set of services including data management, sharing, integrity, availability, as well as high performance and portability.
- The Repository, which provides centralized management for all information used in the computing system, is based on CA's advanced dictionary technology.



Figure 5: Distributed
Processing Services layer

"As we acquire technology, we look at it to see how it best fits into the layers and how it can provide services we don't already have."



"Before you write the software, think about how the pieces are going to connect."

Distributed Processing Services

This layer, shown in Figure 5, insulates CA applications from networking interfaces and protocols. Networking standards such as LU6.2 or DECnet are coded once within this layer, and CA applications are presented with a single networking interface. Supporting all forms of distributed processing across PC LANs, workstations, midrange, and mainframe systems, this layer consists of:

- Cooperative Processing, which lets CA applications cooperate with each other and distribute processing to the most appropriate system.
- Distributed Database Management Services, which let applications access data regardless of physical location.
- The Database Server, which supports centralized databases and client/server implementations. The server manages traffic and access and ensures data integrity.
- The Common Communication Interface, which insulates applications from a variety of networking and communication protocols.

PORTING TO OS/2 2.0

CA watched the OS/2 operating system from its inception, waiting until it was accepted by CA's client base before deciding to port its applications to OS/2 2.0. Three months after OS/2 2.0's March 1992 release, CA announced plans to port seven major applications to OS/2 2.0.

Porting applications is easier for CA than for many software vendors because of the CA90s software architecture. From the beginning, applications are designed with porting in mind. Additionally, all CA developers have experience programming on more than one platform and porting from platform to platform.

The general methodology is to perform a basic Windows-to-OS/2 2.0 port, then enhance the result, adding OS/2-specific features such as multithreading, SOM, and Workplace Shell support. CA divided porting tasks among three groups of developers, reflecting CA90s' division of software architecture:

- Common Services. The CA90s software architecture supports multiple-platform applications by moving as much code as possible into a group of central services that can be used by applications across the product line, as with common memory management functions. The developers in this group worked on porting the Common Services code and converting as much code as possible into SOM/Workplace Shell objects.
- Porting. The developers in this group do a basic Windows-to-OS/2 port, changing as little of an application as possible. These ports result in 32-bit applications that do not incorporate unique features of OS/2 2.0 or the Workplace Shell. As soon as one application is finished, team members move on to the next.
- Product Groups. These teams are responsible for each product; they take the basic 32-bit OS/2 2.0 code from the porting group and enhance it to utilize OS/2 2.0 features such as multithreading. Product groups also add Workplace Shell support and turn applications into objects.

The teams' common goal is to create a 32-bit multithreaded Workplace Shell object that uses interprocess communications and a flat memory model for each application.

While the basic port team uses UNIX-type tools and filters such as *awk* and *sed* to automate the basic port as much as possible, it is not a straight API-to-API port. Some tasks that take many calls in Windows can be done with one or two calls in OS/2 2.0; other simple tasks in Windows become more complicated in OS/2 2.0. In addition, many features, such as memory management, are completely different under the two systems.

SCHEDULING

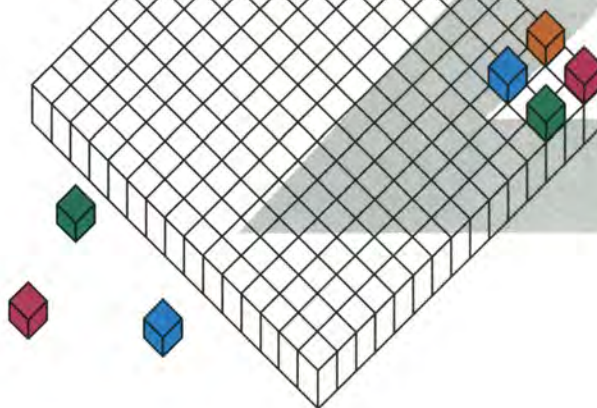
When CA announced plans in June 1992 to port seven applications to OS/2 2.0, they also committed themselves to making demos available for the Fall COMDEX conference in Las Vegas, Nevada. This gave them only five months to learn OS/2 2.0 and port the code.

As the development team had never worked on the operating system, team members first

**Now
\$520 Off**

the purchase of all 3 products until April 6/93.
Offer covers C Set/2, WorkFrame/2 and
Toolkit when purchased together.
List \$995.00. Dealer price may vary.

Work Set/2



Powerful New Tools for OS/2 Programming

The WorkFrame/2 product ... because the best environment for application development is the one you create yourself.

With WorkFrame/2, you can integrate your choice of development tools – including those for DOS and Windows. It's open, configurable and language independent. And it's easy to customize the WorkFrame/2 interface to create your own development environment.

The concept is simple. WorkFrame/2 organizes files into logical units called projects. By associating each project with your personal choice of compiler/debugger/linker/editor you can get the greatest productivity possible from all your development tools.

The C Set/2 product ... because application development should be fast – and simple!

C Set/2 delivers a one-two punch to help you create some of the fastest-performing OS/2-based applications possible.

First, the 32-bit C compiler enables your applications to exploit the speed and power of 386- and 486-based computers. It's the best high-performance code optimizer in the business. With the C Set/2 compiler, unsafe optimizations simply don't exist.

Second, C Set/2 comes with a fully interactive, full-function, source-level 32-bit Presentation Manager debugger. Just point your mouse and shoot, using the graphical PM user interface – or use the keyboard.

Either way it's easy. And you'll get instant feedback on the screen to verify what you're doing. Debugging has never been so simple!

And there's more. The C Set/2 compiler conforms to industry standards – including ANSI C and ISO/IEC – and offers Microsoft C compatibility. With features like a full suite of run-time libraries and 32-16 bit linkage, you can be sure C Set/2 will provide the function and flexibility you need to make application development fast and simple – the way it should be.

OS/2 2.0 Developer's Toolkit ... because it takes the right set of tools to build powerful applications.

OS/2 2.0 Developer's Toolkit is the perfect companion to use with C Set/2. It contains a variety of language-independent application build and productivity tools. For the C Set/2 compiler, Toolkit provides the system linker and system header files. It also contains the import libraries and the NMAKE utility you need to dramatically increase the capabilities of C Set/2 to build powerful applications.

To order or get more information on how IBM application development tools can work in your OS/2 environment,

in the USA call 1-800-342-6672
in Canada call 1-800-465-7999

Making good things happen in application development...





The CA team (left to right): Steve Rainess, Gregory Simeone, Danny Tom, Alex Arthur, Neil Goddard, Bar Tarooqi, Patrick Howard, Ranjit Ramakrishnan, Gary Shattery, John Tillman, and Jacques Leisy

had to train on OS/2 itself. For the most part, members were self-taught: CA provided disks and books and turned them loose. The company's confidence was justified; in under three months, CA had a complete team of OS/2 2.0 programmers.

- Break the application into system-dependent code and application code. As much as possible, minimize the system-dependent code and maximize the application code.
- Use developers with experience working on several platforms and porting applications from one platform to another.
- "Have a lot of patience."

One developer had this to add: "Get lots of sleep ahead of time and hire someone to cut your grass."

CA'S OPINIONS OF OS/2 2.0

Gary Slattery, Computer Associates' vice president of research and development, applauds, "IBM has finally arrived with OS/2 2.0." The developers applauded the 32-bit flat memory model. Assistant vice president for research and development Patrick Howard put it succinctly: "You feel like you are back on a real operating system again."

OS/2's 32-bit performance and protection also received accolades. "One false move doesn't bring the whole system down," says developer Steve Rainess. Developers also found the OS/2 2.0 API well thought out. Developer Danny

CA found that it takes an average of 2 1/2 months per product for a straight port, with the time required dependent largely on the size of the program. Other important factors include the time it takes to translate Windows-specific features and to port the segmented memory management code to the OS/2 2.0 flat memory model.

This speed was not accomplished without a great deal of work. For the transfer to OS/2 2.0, most of CA's team worked 65 to 75 hour weeks, with seven 12-to-16-hour days a week the norm. Developers laughingly made comments like "I think I've got a wife" and "I took OS/2 2.0 on a laptop to the hospital for the birth of my daughter."

CA'S ADVICE FOR PORTING APPLICATIONS

CA has some advice for those considering porting Windows applications to OS/2 2.0:

"Get lots of sleep ahead of time and hire someone to cut your grass."

Tom elaborates, "The API is a lot tighter; things aren't scattered around. If you know what you're looking for, you find it where you expect it. It's clearer how things work.... Concepts are clearer."

The CA developers appreciated the Workplace Shell and the SOM programming model. Developer Jacques Leisy calls the Workplace Shell "another step up from Windows and the Presentation Manager™ in the evolution of graphical user interfaces." Both Leisy and teammate Neal Goddard agree that the object-oriented paradigm used in SOM and the Workplace Shell helped them move code from the application code to the Common Services code.

Many CA developers think the OS/2 2.0 versions of their applications will be better than the Windows versions for several reasons. "First of all, you're not shipping applications, you're shipping objects," says Leisy. Satisfied developers also mentioned OS/2 2.0's superior

performance and the addition of drag-and-drop capability within the application. Overall, CA developers are enthusiastic about OS/2 2.0 and about developing applications for the new operating system. Concludes Leisy, "After people work on OS/2 2.0 for a while, they tend to fall in love with it."

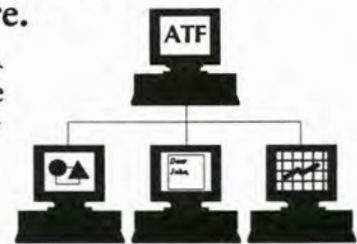
Eric Nelson, IBM Corp., Information Development, 1000 N.W. 51st St., Boca Raton, Fla., 33431. Nelson has worked for IBM for almost two years and is currently working on the OS/2 Toolkit and Technical Library. This is his first article for OS/2 Developer. Nelson holds a B.A. from the University of Texas.



The world's best software for testing networked PM applications now stands alone.

Testing that keeps up with development. Higher quality software.

That's why hundreds of QA engineers and developers use the Automated Test Facility to test their networked Presentation Manager and Windows apps.



ATF NetWorked

Both ATF NetWorked and Softbridge's new ATF WorkStation were built to stand up to the complexities of testing a graphical user interface like PM.



ATF WorkStation

To find out how ATF can help you meet the challenges of testing PM apps, call: 800-955-9190.



Softbridge, Inc.
125 CambridgePark Dr.
Cambridge, MA 02140
617-576-2257 (Phone)
617-864-7747 (FAX)



32-Bit

Designing An OS/2 Device Driver

by Steven Mastrianni

The following is an excerpt from the book *Writing OS/2 2.0 Device Drivers in C*, reprinted with permission of Van Nostrand Reinhold.

To successfully design an OS/2 device driver, a developer must understand the device driver's role and have a solid working knowledge of the OS/2 operating system and design philosophy. Debugging a device driver can be difficult, even with the proper tools. The OS/2 device driver operates at Ring 0 with full access to the system hardware. However, it has access to almost no OS/2 support services except for a handful of *DevHlp* routines. As many device driver failures occur in a real-time context (for example, during interrupt handling), it may be difficult or impossible to find a problem using normal debugging techniques. In such cases it is necessary to visualize the operation of the device driver and OS/2 at the time of the error to locate the problem.

THE BASICS OF DRIVER DESIGN

The device driver receives two basic types of requests, those that can be completed immediately and those that cannot. It receives these requests via a standard data structure called a request packet. The device driver manipulates request packets with the Device Helper or *DevHlp* routines, as in Figure 1.

Requests that can be completed immediately are handled as they come in and sent back to the requestor. Those that cannot be handled immediately, such as disk seeks, are queued for later dispatch by the device driver. Disk device drivers usually sort pending requests for disk seeks in sector order, thereby minimizing disk seek time.

The OS/2 device driver plays an additional role in system performance and operation. When a device driver is called to perform a request that cannot be completed immediately, it **B**locks the requesting thread, relinquishing the CPU and allowing other threads to run. When the running request is completed, the thread is immediately **U**n**B**locked and run. The device driver then queries the request queue for any pending requests that may have come in while the thread was blocked. When an application calls a device driver, the application's local descriptor table (LDT) is directly accessible by the device driver.

There are two types of OS/2 device drivers, block and character. Block device drivers are used for mass storage devices such as disks and tape devices. A character device driver such as a modem or serial terminal is used for devices that handle data one character at a time. OS/2 device drivers can support multiple devices, such as a serial communications adapter with four channels or a disk device driver that supports multiple drives.

OS/2 device drivers receive requests from the OS/2 kernel on behalf of an application program thread. When the device driver is originally opened with a *DosOpen* API call, the kernel returns a handle to the thread that requested access to the device driver. This

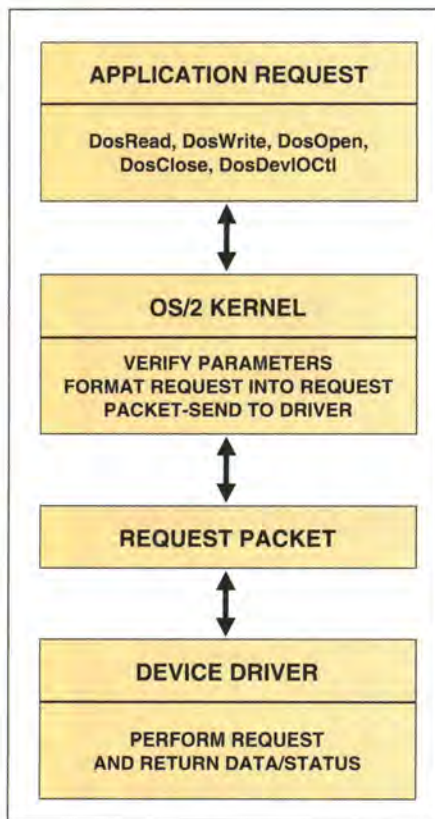


Figure 1: Application-to-device driver interface

Power tools that give you more

Introducing IBM Tools Catalog



Order direct.
1 800 IBM-CALL

IBM

More Power.

**IBM Extended
Services
for OS/2**

Extend your reach.

Tap into the power of OS/2® with flexible communications, versatile connections and a sophisticated database. Extended Services enables you to view your data, communicate with outside information services and record transactions all at the same time.

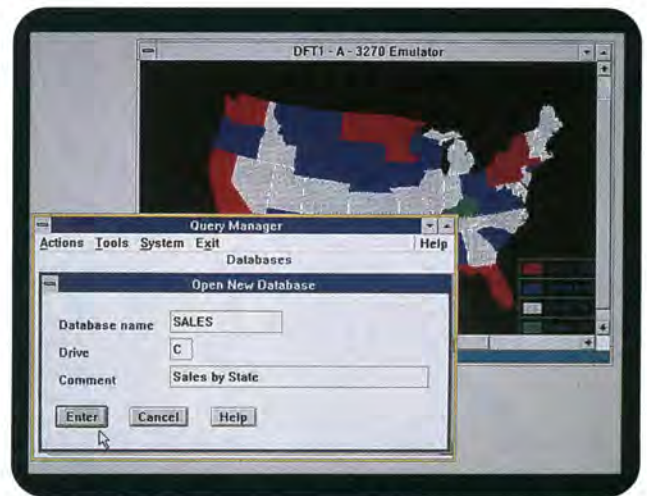
Together Communications Manager, Database Manager and Query Manager make up the components of Extended Services. Distribute your data and processing tasks among workstations, departmental systems and hosts. The 3270 EHLLAPI interface now allows multiple, simultaneous access by applications; these functions can now be used in REXX Procedure Language Programs for 3270 and 5250 applications. Virtual Device Drivers (VDDs) allow NETBIOS applications, running in DOS partitions, to share an adapter with other DOS and OS/2 NETBIOS applications. Existing DOS 3270 EHLLAPI applications can run in a DOS partition, sharing Communications Manager EHLLAPI with other DOS and OS/2 applications.

Enhancements to the Communications Manager include improved APPC support — with simplified configuration and improved performance, most notably on a Local Area Network (LAN). You can now use long file names for file transfers! You'll see an increased number of simultaneously active workstations through a SNA gateway and better SNA gateway LU pooling to improve session accessibility.

The Database Manager allows existing DOS and DOS Windows™-based machines to become database clients on a LAN. Enhanced SQL capabilities bring the database into closer compliance with SAA™ SQL Level I. Roll-forward recovery (archive logging) reduces the need for manual recovery if your system fails.

OS/2 Extended Services lets you issue SQL statements, database environmental commands and utilities directly from the OS/2 command line. You'll access, update and administer data without having to learn the SQL syntax, or download, import and convert DB2® and SQL/DS™ host data. You can also use the comprehensive set of database administration tools and facilities, including roll-forward recovery and First Failure Support Technology/2 (FFST/2). OS/2 ES lets you log error information needed for problem resolution, helping you to minimize the possibility of running into the same error again.

- Contains integrated communications and database manager functions of OS/2 EE and of SAA Networking Services/2, helping you manage information and move it with ease.
- Provides multiuser database functions with Database Server for OS/2.
- Extends communications and database functions to selected non-IBM hardware platforms and to selected, compatible OS/2 operating systems.
- Provides multinational solutions for your expanding global business needs with twelve versions including U.S. and U.K. English.
- IBM SAA Distributed Database Connection Services/2 Program allows a Database Manager client to access DB2, SQL/DS and OS/400® databases on S/370, S/390™ and IBM midrange AS/400® computers.



With OS/2 ES you get industrial-strength function supported by a rich array of tools.

To help better serve your needs seven new productivity aids are shipped with Extended Services. Included are print and file transfer utilities, a keyboard utility, a problem determination tool, APL support and a database performance optimization tool. A new publication describing these applications is shipped with Extended Services.

Would you like to order OS/2 2.0?

Here's how! Call:

Corporate accounts and invoice orders	1 800 IBM-CALL
Credit card orders	1 800 3IBM OS2
In Canada	1 800 465-7999



Notebook control, one of seven tools within CUA Control Library /2.

IBM C Developer's WorkSet/2

The right tool for your job.

The OS/2 2.0 Application Development Tools can help to make your programmers more efficient, more precise and more productive. These tools work well individually or together to provide a complete 32-bit C-language environment so you can develop a full range of applications — from retail to mission-critical. For the greatest convenience choose a packaged solution. The IBM C Developer's WorkSet/2 kit bundles the Developer's Toolkit and WorkFrame/2 with the C Set/2 — providing a total application development package for OS/2.

- Takes advantage of the power of 80386- and 80486-based system units through state-of-the-art code optimization.
- The OS/2 2.0 Developer's Toolkit provides a comprehensive set of language-independent tools and code samples fundamental to building advanced OS/2 2.0 applications.
- IBM OS/2 2.0 Technical Library supplements the Toolkit with programming guides and other printed references.
- IBM WorkFrame/2 makes application development easier by organizing projects and allowing you to integrate your preferred tools.
- IBM C Set/2 includes a 32-bit SAA C Compiler with its runtime libraries and a fully integrated, source level Presentation Manager™ debugger.

IBM Common User Access Controls Library/2

Get a leg up on OS/2 code.

Save your energy for the really creative parts of your work and let one of IBM's newest products, CUA Control Library/2, provide the interface for GUIs. Whether you're writing for Windows or OS/2 applications, it's all right here for you.

- Offers ability to conform to CUA 91 architecture for OS/2 V1.3 and Windows V3.0 and 3.1.
- Provides for easier application migration to OS/2 V2.0 because the CUA Controls APIs for OS/2 V1.3 and Windows V3.0 and 3.1 are consistent with those found in OS/2 V2.0.
- From containers to sliders, files to notebooks, fonts to value sets, CUA Controls Library/2 does it all.

Ordering Information

Program No.		OTC
5871-AAA	Extended Services for OS/2	
F/C 2778	ES for OS/2	\$595
F/C 2780	ES for OS/2 with Administrator's Kit	\$760
F/C 2278	ES with Database Server for OS/2	\$1,995
F/C 2285	Database Client Feature	\$75
5871-AAA	Distributed Database Connection Services/2	
F/C 2304	Single-user:	\$500
F/C 2310	Multi-user:	\$4,680
5871-AAA	OS/2 2.0 Tools for Application Development	
F/C 2398	C Developer's WorkSet/2	NOW \$375! \$895
F/C 2394	C Set/2 Compiler and Libraries	\$696
F/C 2396	OS/2 2.0 Developer's Workbench	\$199
F/C 2390	OS/2 2.0 Developer's Toolkit	\$119
F/C 2392	IBM WorkFrame/2	\$90
5876-XXX	OS/2 2.0 Technical Library	
F/C 2700	OS/2 Technical Library	\$299
5871-AAA		
F/C 2463	Common User Access Controls Library/2	\$624

63% savings!

Get C Developer's WorkSet/2 by
April 6, 1993 for only \$375.

IBM
AD/Cycle
Prolog/2

IBM SAA AD/Cycle Prolog/2 is a high-productivity, logic-based programming language. With its powerful logic expressions and built-in control flow constructs, the technology Prolog/2 is ideally suited for complex logic applications. Applications developed using Prolog/2 can easily be ported to the IBM SAA environment.

- IBM
AD/Cycle
PL/I
Package/2

With the new IBM SAA AD/Cycle PL/I Package/2 you can develop applications targeted for a S/370 host or OS/2. You can realize the productivity and lower cost benefits of a program developed in an OS/2 environment, editing and compiling. It is a common PL/I development environment on OS/2.

Local Area Network (LAN)-based applications, linking multiple PC workstations. Applications can be readily ported to OS/2.

Skills in PL/I can now be carried to the OS/2 environment.



APL2™ is an extremely effective programming language in a wide range of uses from interactive computing to "what if" modeling, exploratory programming and application program design.

Completely revised for 1992, APL2/PC provides high compatibility with the 370/390 mainframe and RISC System/6000™ APL2 products. APL2/PC permits transparent import/export of programs and data via the IN and OUT commands, and allows for a high degree of connectivity with APL2 on the host and RISC platforms.

- 4 Order direct, 1 800 IBM-CALL.

IBM Workstation Interactive Test Tool

Fast, flexible testing.

The Workstation Interactive Test Tool easily automates many of the tasks

associated with interactive application testing. It's cost-effective, too! Customers have realized 50-70 percent savings in regression test time.

- Test both host- and workstation-based applications.
- Create test cases as you go, recording keystrokes or mouse movements.
- Save and compare screens and/or areas within screens.
- Use your own editor to tailor already recorded test cases.
- Add logic to your test cases by imbedding Procedures Language 2/REXX commands.
- Watch your test cases playback or run them at night on your display in unattended batch mode.
- Measure performance of key functions by inserting time stamps.

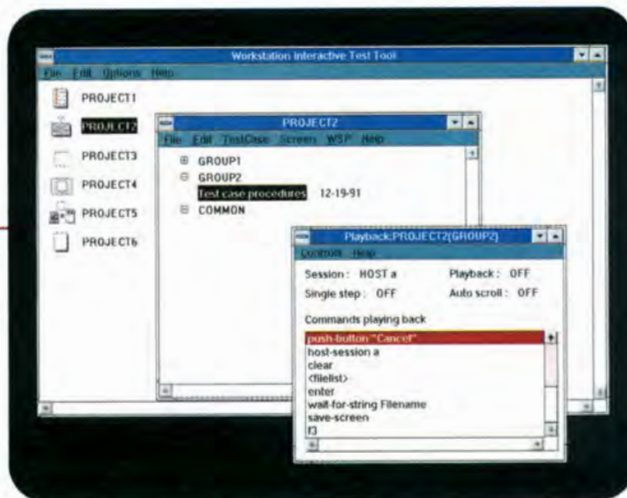
IBM Software Analysis Test Tool

Improved quality. Increased productivity.

Improve your application development productivity with the Software Analysis Test Tool. This tool helps you — the developers, testers and maintainers — discover how well your application has been tested.

Software Analysis Test Tool is a key component in improving application development productivity. It provides complete testing analysis, graphical display of test case coverage, animation of test case execution and online documentation.

- Provides a graphical display of the execution flow of your program.
- Offers a selection of program metrics to evaluate testing coverage.
- Supports IBM languages in the runtime environment.
- Supports interactive browsing, query and reporting of test coverage information.
- Helps you test, maintain, measure and analyze the quality of existing and newly produced software with graphical displays for a clean system and structure view.



Create, analyze and playback test cases/scripts automatically with Workstation Interactive Test Tool.

IBM AD/Cycle Translation- Manager/2

It's all in the translation.

IBM SAA AD/Cycle Translation-Manager/2 helps you, as a professional or occasional translator, translate from a selected national language into another national language. It allows you to address translation more effectively in the early stages of application development. Regardless of the size of your environment, TranslationManager/2 will support you through the entire application development process, starting with terminology created at the requirements gathering stage, through translation of documentation and program-integrated text, and maintenance of this textual information.

- Provides enhanced business solutions by support of translation, as an integral part of developing an application, or in standalone mode outside of the AD/Cycle environment.
- Increases productivity with a variety of convenient and powerful translation, dictionary and linguistic support functions.
- Reduces systems management complexity and increases operational productivity of the total translation process.
- Integrates previously created translation and dictionary files. Translations obtained previously by using TranslationManager/2 can be reused and adapted through the Translation Memory function.

Ordering Information

Program No.		OTC
5696-308	AD/Cycle Prolog/2	
F/C 0980	Runtime Facility	\$420
F/C 0995	Development Facility Feature	\$5,250
F/C 1016	Runtime Extension Feature	\$336
F/C 1018	Development Facility Extension Feature	\$4,200
5601-388	PL/I Package/2	\$2,750
5799-PGG	APL2 for the IBM PC	\$630
5601-442	Software Analysis Test Tool	\$6,305
5601-441	Workstation Interactive Test Tool	\$2,520
5621-327	TranslationManager/2	\$5,250

More Action.

IBM
CICS
OS/2

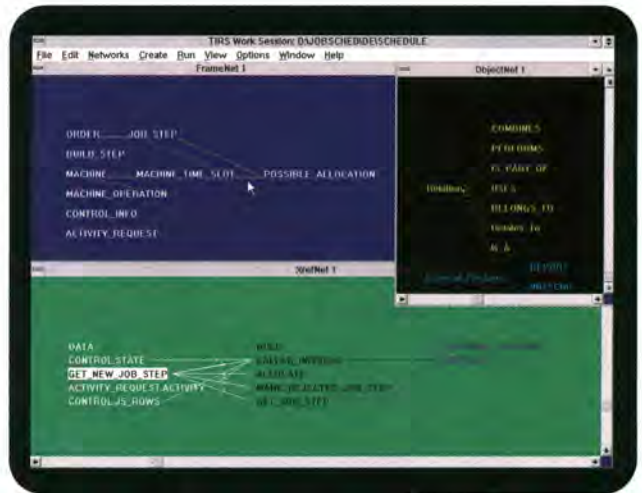
Now you've got local-motion.

Turn your PS/2 or PC into an application powerhouse with CICS OS/2™ V1.2.

CICS OS/2 lets you use the programming workstation as a development platform, bringing flexible scheduling to your department while capitalizing on existing CICS programming skills. This powerful environment offers the portable EXEC CICS command level Application Programming Interface in the COBOL and C programming languages and allows you to develop and test your applications. Many production applications can be produced in their entirety on a standalone workstation.

- CICS OS/2 provides local transaction processing with or without host attachment.
- Expanded communication support for SAA allows you to capitalize on your investment in OS/2 EE.
- Developers can take advantage of techniques like Advanced Program-to-Program Communications (APPC) without learning APPC coding.

- Offers automatic transfer of applications from workstation to host.
- Works under OS/2 and C Set/2.



TIRS provides an interactive windowed graphic interface for creation and maintenance of knowledge bases.

The
Integrated
Reasoning
Shell

Platforms at your request.

Enhance your applications with expertise using The Integrated Reasoning Shell® (TIRS™), a tool that provides application developers with a productive development environment and high performing, compiled applications. Multiplatform support provides you with the flexibility to deliver applications where they are needed.

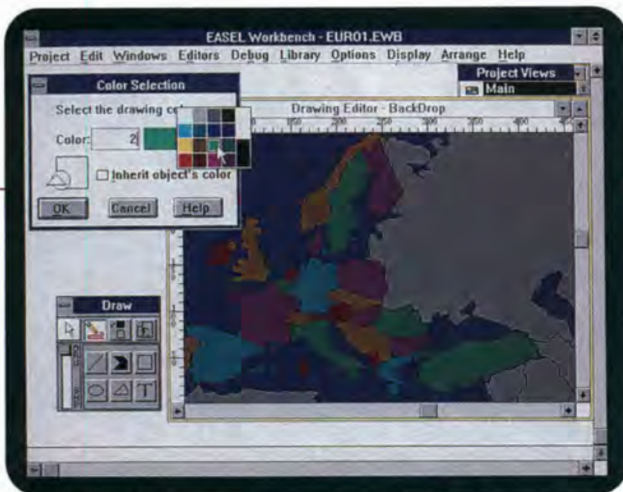
- Provides advanced artificial intelligence (AI) function for development of many business application types.
- Includes a powerful Knowledge Representation Language.
- Offers compiled applications for fast runtime performance.
- Provides portability to multiple platforms, including selected SAA environments for flexible growth.
- Enhances developer productivity with mouse-assisted, graphical development interface under OS/2.
- Improves performance using a runtime subsystem architecture for CICS and IMS.
- Uses direct interlanguage communication.
- Includes sample knowledge bases to assist application developers in learning TIRS.

IBM
CSP/2
Application
Development

Define, test and view.

As part of the IBM SAA Cross System Product Set, CSP/2AD Version 1 helps you develop high-quality applications faster and easier. CSP/2AD runs under OS/2 and provides a highly productive environment for defining and testing Cross System Product applications for OS/2 and MVS. It has an easy-to-use graphical interface that conforms to the Common User Access architecture. Application components can be viewed, accessed and modified using a graphical representation of the application, simplifying both development and maintenance. Finally, you can use the interactive test facility to completely test an application on the workstation prior to generation. Additional features include:

- Statement templates and interactive syntax checking.
- The ability to monitor application execution during testing, watch and modify data values, set breakpoints, modify application logic and continue testing.
- Source level debugging.
- Utilities to transfer applications to the IBM SAA Cross System Product/370 Application Development product for generation.



Enhanced graphical applications are yours with EASEL Workbench.

**EASEL
Workbench**

Powerful application development.

Easel Corporation's EASEL Workbench® provides a robust set of tools for developing client/server applications. Easel customers are achieving success by using EASEL Workbench to rapidly prototype and deliver a wide variety of sophisticated business solutions for OS/2, Windows and DOS. A variety of client/server architectures are supported by EASEL Workbench applications, including transaction processing and DSS applications that access local and enterprise data via SQL or other methods.

- Lets developers build SAA/CUA™ applications.
- Provides key component for AD/Cycle Analysis/Design and Generator strategies.
- Leverages power of OS/2 development platform.
- Compiles and executes for OS/2, Windows and DOS environments.
- Provides incremental compiler, source level debugger and trace facilities and organization tools to manage development environment. **CSP**

**InfoSpan
Repository
Management
System**

A server-based repository.

The InfoSpan™ Repository Management System (IRMS™) by InfoSpan Corporation is the industry's first ANSI/FIPS Information Resource Dictionary System (IRDS) conforming repository. IRMS is a comprehensive LAN server-based system. IRMS is the foundation for InfoSpan's CASE and Reverse Engineering Tool Integration solutions. It is enabled to IBM's AD/Cycle

Platform Services and Information Model. Subsets of IBM's Information Model are offered on IRMS.

- Offers a single design with a common user interface, single-user and multiple-user versions.
- Includes tool services to build custom data dictionaries to administer data and standardize data elements.
- Provides solutions for information resource and asset management, software design and change management.
- Supports standardization of data elements that need aliasing, data attributes and approval management, all by means of custom naming policies.

IRMS Repository Management System tools include Repository Manager, IRMS Repository Modeler & Prototyper, Graphical Repository Browser & Navigator, Reporters & Analyzers, Graphical Repository Administrator, Data Element Policy Defines and Administrator, and Repository Loader. **CSP**

Ordering Information

Program No.	OTC
5688-101 CICS OS/2	\$810
The Integrated Reasoning Shell	
5621-003 TIRS Development/2	\$8,595
5621-004 TIRS Runtime/2	\$859
5688-205 CSP/2AD Version 1	\$2,520
EASEL Workbench Tools	
5758-EC0 EASEL/2 Workbench with Production Compiler	\$11,900
5758-EC1 EASEL/2 Workbench Migration from IBM EASEL	\$2,800
5758-EC4 Additional EASEL/2 Production Compiler	\$5,900
5758-EC7 EASEL/2 Runtime System	Call
InfoSpan Repository Management System*	
5758-IFC Repository Manager	\$13,200
5758-IFD Modeler & Prototyper	\$2,750
5758-IFE Browser & Navigator Manager	\$1,750
5758-IFF Reporter & Analyzer	\$2,250
5758-IFG Graphical Reporter & Analyzer	\$2,500
5758-IFH Graphical Repository Administrator	\$3,450
5758-IFI Name Administrator	\$4,450
5758-IFL Repository Loader	\$4,600

* Quantity discounts and multiuser package prices are available.

Save up to 20%
on additional licenses.

More Energy.

Micro Focus COBOL Workbench

This workbench comes with a toolset.

Micro Focus® COBOL™ Workbench is an integrated collection of programming productivity tools that complement the Micro Focus COBOL compiler. It provides a highly integrated and responsive development environment that can dramatically improve programmer productivity. The function and design of the COBOL Workbench environment allows you to quickly address your application backlog, produce efficient and higher quality applications and effectively maintain COBOL applications on a single PS/2. These applications can be targeted to a wide range of host and PC environments. Key functionality in the Micro Focus Workbench includes:

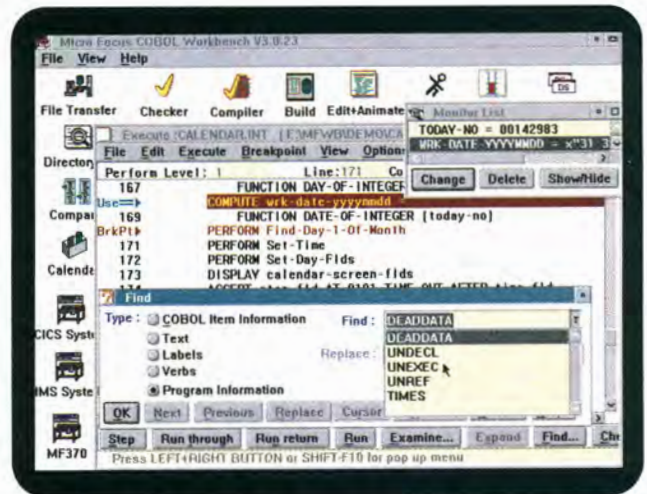
- Full-screen text editor
- Mainframe COBOL Compatibility (IBM OSVS, VSC2, DOSVS, COBOL 370)
- EBCDIC data file support
- Source Code Level debugging with ANIMATOR.

The cost benefits of using the Micro Focus Workbench for mainframe development are:

- Consistently fast response times
- Saves mainframe cpu cycles
- Reduced TSO charges
- Dedicated development environment for the programmer on the PS/2.

Add-on products that complement the Micro Focus COBOL Workbench:

- Micro Focus CICS Option™ provides mainframe CICS Command level compatible application development environment.
- Micro Focus CICS OS/2 Option™ is a full-function IBM CICS multiuser, multitasking environment.
- Micro Focus IMS Option™ provides a mainframe compatible IBM/DB and DC development environment.
- Micro Focus IMS Production Option™ is a multiuser execution environment.
- Micro Focus 370 Assembler emulates 370 mainframe assembler on the PC. [CSP](#)



Micro Focus COBOL Workbench supports effective and efficient development that can improve productivity.

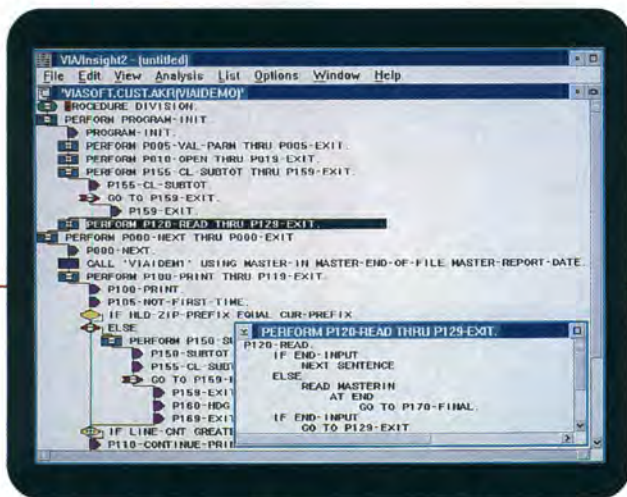
BACHMAN Development Tools

A model performance.

This easy-to-use product set by Bachman Information Systems, Inc. provides you with the ability to do model-driven development either of new applications or from your existing investment in database designs. With Bachman tools both analysts and programmers will see a decrease in their work effort as well as an improvement in application quality.

- BACHMAN/Analyst™ produces models that can be forward engineered to production-quality DB2 or SQL/DS systems.
- BACHMAN/Capture™ products allow IS departments to take full advantage of data designs residing in existing database structures.
- BACHMAN/Database Administrator™ (DBA) optimizes large DB2 designs and DB2 objects. It provides re-engineering capabilities with business models.
- BACHMAN/Designer™ generates external source format optimized for IBM's Cross System Product application generator.
- BACHMAN/Shared Work Manager allows many analysts to work concurrently on an enterprise model. It lets you create subset assignments for different analysts, monitor and view their work in progress and reassembles their submodels into a finished enterprise model.
- BACHMAN/DBA Enabler allows the database administrator to design better performing databases for SQL/DS.

[CSP](#)



Redevelop your aging, complex systems into modern, leading-edge systems with VIASOFT's Existing Systems Workbench.

VIASOFT Existing Systems Workbench

Put new life in old programs.

IS professionals, empower yourselves with an integrated set of tools that effectively enhances, maintains and redevelops aging applications. Using VIASOFT, Inc.'s Existing Systems Workbench management you can gain control of existing systems; programmers can understand and improve aging, complex systems; and analysts can extract business rules from existing applications and reuse them in new systems. The OS/2 workstation tools mentioned below work in conjunction with their respective host products.

- Using VIA/Insight 2 and VIA/SmartDOC2, application programmers can significantly decrease the amount of time they spend understanding the legacy code of the business enterprise.
- Using VIA/Renaissance2, programmers can turn large programs into smaller, easier to manage and understand module programs.
- Using VIA/ReCap2, programmers can quickly measure the complexity of an application, including Function Point measurement, each time the application is changed. **CSP**

Application Development Workbench for OS/2

Solutions today.

KnowledgeWare® builds, markets and supports CASE tools for developing high-quality, well-documented business applications for multiple platforms.

KnowledgeWare's extended family of products, including the Application Development Workbench (ADW), maintains and enhances current software applications.

ADW is composed of a highly integrated, graphics-based set of workstation software, supplemented by optional mainframe-based tools. The ADW products aid users in all phases of application development:

- Planning and Analysis
- Prototyping
- Design
- Code generation
- Systems documentation
- Maintenance.

ADW lets users put the focus on defining and solving organizational needs. For instance, developers can gather application requirements independently, then generate code for applications running on mainframe, midrange and PC hardware. This integration strategy means developers can use and reuse design logic across all platforms.

The family of ADW products includes ADW/Planning Workstation, ADW/Analysis Workstation, ADW/RAD Workstation, ADW/Design Workstation, ADW/CSP Enablement Facility and ADW Documentation Workstation.

CSP

Ordering Information

Program No.	OTC
Micro Focus COBOL Workbench	
5758-MFF Micro Focus COBOL	\$750
5758-MFH CICS OS/2 Option	\$1,250
5758-MFI CICS Option	\$1,250
5758-MFK IMS Option	\$1,250
5758-MFM 370 Assembler	\$1,250
5758-MFA COBOL Workbench	\$2,500
Complementary Micro Focus COBOL Workbench products	
5758-MFB CICS OS/2 Option	\$3,500
5758-MFC CICS Option	\$3,500
5758-MFD IMS Option	\$3,500
5758-MFE CICS and IMS Option	\$4,500
BACHMAN Development Tools	
5758-BAA Analyst	\$10,000
5758-BAB Capture for IMS	\$10,000
5758-BAC Capture for COBOL	\$5,000
5758-BAD Capture for PL/I	\$2,500
5758-BAM DBA	\$15,000
5758-BAU Designer	\$10,000
5758-BAV DBA Enabler	\$15,000
5758-BAW Shared Work Manager	\$5,000
VIASOFT Existing Systems Workbench	
5758-VSH VIA/Insight2	\$2,500
5758-VSJ VIA/Renaissance2	\$3,000
5758-VSL VIA/Recap2	\$1,500
5758-VSN VIA/SmartDoc2	\$1,000
Application Development Workbench	
5758-K31 ADW/Planning Workstation	\$10,750
5758-K32 ADW/Analysis Workstation	\$10,750
5758-K33 ADW/Design Workstation	\$10,750
5758-K34 ADW/RAD Workstation	\$10,750
5758-K35 ADW/Documentation Workstation	\$10,750
5758-K26 ADW/CSP Enablement Facility	\$1,100
5758-K30 ADW/REF	\$25,000
5758-K36 ADW Starter Kit*	\$24,500

*One each: Planning, Analysis, Design, CSP Enablement Facility, RAD and DOC Workstations. Available to new accounts only. Limit one per account.

More Success.

IBM OS/2 LAN Server 3.0

Good news travels fast.

Why settle for software that just lets you share files, printers and applications? IBM's OS/2 LAN Server 3.0 includes new functions that provide greater reliability, faster access to data and increased security for your resources and data.

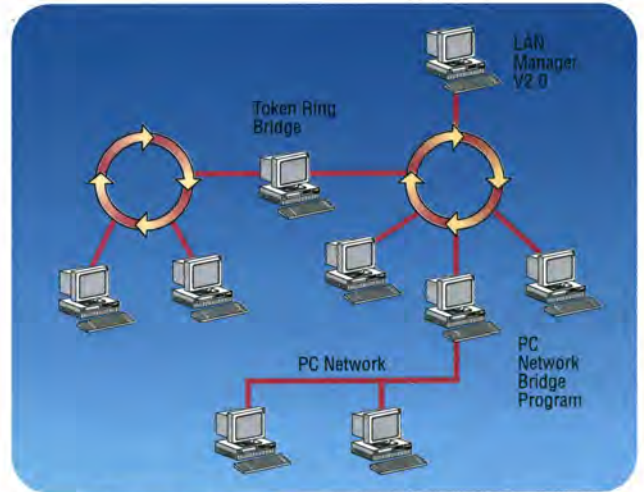
With OS/2 LAN Server 3.0, DOS (including DOS/Windows support) and OS/2 requesters can easily access LAN resources, no matter where they are located. Even remote resources appear to reside on end-user workstations. This single-system image is achieved through advanced administrative facilities called Domains and Aliases. They allow users to access resources that reside on other OS/2 servers — or even Microsoft® LAN Manager servers — with a single network logon.

As a result, users can transparently access and share directories, files, disks, printers and programs, plus serially attached devices such as plotters, modems and scanners. Also, administrators can more easily make changes in resource locations, security rights and user accounts.

OS/2 LAN Server 3.0 is available in two configurations — Entry and Advanced.

- Brings you High Performance File System (HPFS) for i386™ and i486™ processors, which speed access to server data.
- Provides support for up to four LAN adapters.
- Includes Remote Initial Program Load (RIPL), which supports diskless DOS and OS/2 workstations, enhances security and allows you to control application levels.
- Gives UPS support to keep the server running in case of power failure.
- User Profile Management (UPM) provides a graphical interface for administration of user's assigned resources.

The LAN Requester distributed feature of OS/2 LAN Server 3.0 gives you the OS/2, DOS and DOS Windows Requesters. Each distributed feature can give you simultaneous access to multiple servers, increasing the flexibility of your network. The OS/2 LAN Server 3.0 package includes a license for one LAN Requester feature. For each additional feature you order, you may make a single copy of the LAN Requester and we will send you a proof of license certificate. This allows you to order just as many LAN Requesters as you need.



Manage token ring and broadband networks with LAN Manager.

IBM LAN Network Manager

The most management for your LAN.

Whether you manage your LANs locally from a workstation or centrally from NetView®, you can do it more efficiently with this LAN management tool. Running alongside your other applications on OS/2 EE, this application allows you to manage multisegment token rings, broadband and baseband PC Networks, as well as IBM LAN Bridges.

If you have a remote single-segment IBM Token-Ring or PC Network and want to integrate it with NetView, IBM LAN Network Entry lets you do it. Running as a task in OS/2 EE, this software uses Communications Manager to communicate with NetView via an existing LAN SNA gateway. It provides a subset of the expanded set of NetView commands supported in IBM LAN Network Manager 1.1.

- Identifies and records network errors, access failures and LAN workstation adapter failures.
- Automatically maintains detailed configuration information about network devices and their locations.
- Communicates with the host via an existing LAN gateway or over a Synchronous Data Link Control (SDCL) link.

IBM LAN to LAN Wide Area Network

From LAN to WAN.

Connect remote LANs to your SNA or X.25 backbone — and expand LAN applications without having to duplicate teleprocessing lines. This program allows IBM Token-Ring Network, IBM PC Network and Ethernet LANs to communicate with each other over a wide area network using NETBIOS to NETBIOS flows. Ideal for small LANs that need access to servers on a small number of "hub" LANs. It will grow with you to support large LANS in complex networks. Anywhere your SNA network goes you can have NETBIOS connectivity available.

- Provides full 802.2 and LU 6.2 support.
- Eliminates need for dedicated processors.
- Sends alerts to a host via LAN Manager.

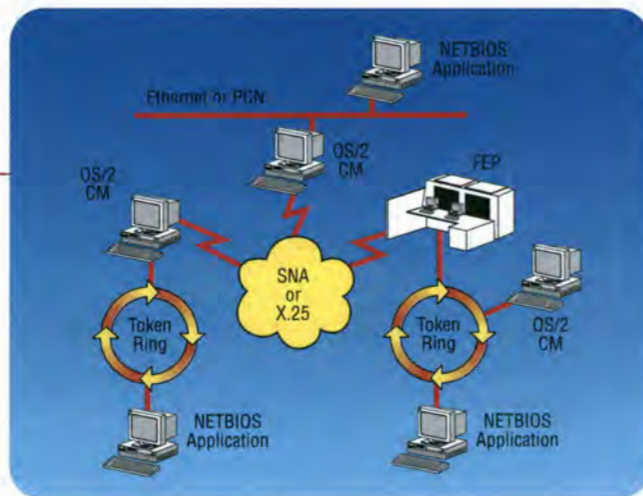
IBM LAN Installation Utility/2

Now installation's easier.

Need help with installation? The IBM LAN Installation Utility/2 (LIU/2) is an aid in the installation of OS/2 2.0, Extended Services 1.0 and LAN Server 2.0. With this program these systems can be installed simultaneously with little effort and time. Working on a redirected drive replication philosophy, the IBM LAN Installation Utility/2 eliminates the need for the insertion of the multiple diskettes images at the target workstation. All of the new operating system and subsystem code resides on a server drive that is accessible to the target machine by a redirection facility.

That's not all — LIU/2 can be used to deliver any customer-defined package of files, including applications, to any target machine.

- Creates a common software image on a source machine, and sends it to the LIU/2 distribution server.
- On the distribution server, builds customized files for each target machine and produces two boot diskettes for each target machine.
- At the target machine, the boot diskettes are loaded and LIU/2 installs OS/2 from the distribution server without any further human intervention.



Anywhere your SNA network goes you can have NETBIOS connectivity.

IBM LAN Management Utilities/2

At your service.

Designed for the enterprise server/requestor environment, this set of OS/2-based services includes applications for configuration, performance and fault management. It also includes the command/data transport applications required to use them.

- A graphical display provides a display of LAN stations and their status, panel-driven commands and notification of alert conditions.
- Includes a transport method by which system management applications can be invoked remotely and system management data can be sent to a central collection point.
- Includes a set of IBM-supplied LAN system management applications and software to maintain an OS/2 database that contains the response information from the IBM-supplied management applications.

Ordering Information

Program No.	OTC
5871-AAA OS/2 LAN Server	
F/C 3721 OS/2 LAN Server 3.0 Entry	\$795
F/C 3733 OS/2 LAN Server 2.0 Advanced	\$2,295
F/C 3764 LAN Requestor distributed feature (with certificate)	\$75
5871-AAA	
F/C 2881 IBM LAN Enabler 2.0	\$100
5871-AAA LAN Network Manager	
F/C 1468 LAN Network Manager V.1.0	\$4,190
F/C 1472 LAN Network Manager V.1.1	\$5,240
F/C 1476 LAN Network Manager Entry	\$1,365
5871-AAA	
F/C 1480 LAN to LAN WAN	\$2,090
5799-PTC IBM LAN Installation Utility/2	\$ 350
5799-PYA LAN Management Utilities/2	\$795

More Access.

IBM Personal Communications /3270

Move up to multiple sessions.

If your users need multiple concurrent host sessions — or multiple DOS sessions with native Windows support — here's your answer. IBM Personal Communications/3270 Version 2.0 provides advanced host communications and gateway services for PS/2s and PCs. It offers both a DOS mode with enhanced workstation function, a Windows mode and a gateway mode all in a single package.

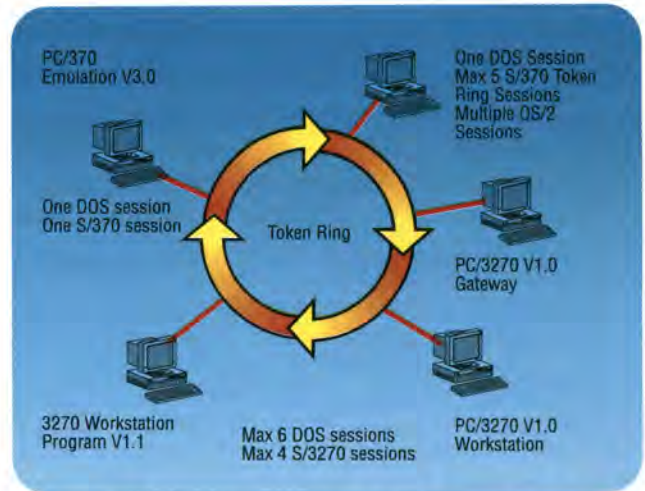
- Host connectivity via SDLC, Token Ring, Ethernet, 3270 Distributed Function Terminal (DFT) and Control Unit Terminal (CUT) and asynchronous connections.
- Provides enhanced fonts for easy-to-read screens.
- Facilitates the management of large installed bases via a quick installation option.
- Allows users to move from DOS to Windows or gateway mode without having to buy an additional package.

IBM Multi- Terminal ASCII Emulator

Emulate ASCII on your PC.

If you're looking for a way for your PS/2 workstations to concurrently emulate multiple ASCII terminals, IBM Multi-Terminal ASCII Emulator/2 is the answer. It allows you to centrally emulate multiple asynchronous terminals on a single workstation — alleviating the need for unique consoles for devices in the system and consolidating their monitoring to a single workstation. You can centralize access to the ASCII operator terminals because of this program's key support of LU6.2 connections.

- Enhances systems management by providing central site access to non-SNA devices and management systems within an SNA network.
- Protects your investment by expanding central site support into the non-SNA portions of the network.
- Enhances network support personnel productivity by improving information via direct access to non-SNA portions of the network.
- Enables growth and migration in a heterogeneous network by allowing management access as the initial step to total management integration.



Move up to multiple-session 3270 emulation for DOS and Windows environments with Personal Communications/3270 V2.0.

IBM Networking Services/ DOS

Protect your investment.

Networking Services/DOS lets you integrate existing DOS and Windows workstations into your computing network. From small workstations to the largest mainframe system, Networking Services/DOS allows DOS and Windows workstations to participate in networks with both IBM and non-IBM systems.

That's not all. Networking Services/DOS lets you bring your DOS workstations into a distributed computing environment. This type of distributed computing combines the control, speed and storage capacity of large systems with the usability and flexibility of inexpensive DOS workstations.

- Provides a low-memory Advanced Program-to-Program Communication solution for DOS and Microsoft Windows environments.
- Allows you to effectively develop and run distributed applications for the DOS and Windows environment, since Networking Services/DOS can run in as little as 90KB of conventional memory, plus the required 50 to 60KB of memory when communication applications are run.
- Uses CPI-C, a consistent programming interface for distributed applications throughout the networks and aids portability of applications and programming skills.
- Provides easy integration into dynamic APPN networks, as well as existing SNA subarea networks.



OS/2 TCP/IP with X-Windows helps you broaden your connections.

IBM TCP/IP Connection Tools

Make the right connection.

Widely used in government, academic, engineering/scientific and commercial environments, TCP/IP protocols allow you to network with the most commonly used workstations and hosts — IBM and non-IBM — including UNIX® systems. Because IBM's family of TCP/IP products is so extensive, users can interoperate with other systems in these networks — whether they are running DOS, Windows, OS/2, OS/400, AIX®, VM™, MVS™ or software from vendors such as Sun®, DEC®, HP® and Apple®.

TCP/IP Version 2.1 for DOS supports TCP/IP in DOS or Windows Mode. TCP/IP for OS/2 supports OS/2 systems and is integrated for use with the Presentation Manager. Both products interoperate with Novell® and IBM LAN requestors so that you can use TCP/IP applications at the same time that you are using your local servers.

Both products support the most popular LAN environments — Ethernet and Token-Ring. They also can be used with serial lines including dial up. The OS/2 product also has an optional kit that supports X.25 networks so that you can use both public or private X.25 networks.

A full suite of TCP/IP applications is included with the base kits for the DOS/Windows and OS/2 products. Each has an optional kit for NFS (Network File System). In addition, both DOS/Windows and OS/2 have programmer's Toolkits that support the most common application interfaces. Applications can be written using either the DOS or OS/2 products that can communicate with either IBM systems such as MVS/CICS or non-IBM systems.

TCP/IP Support

	OS/2	DOS Windows
Applications – Base Kit		
FTP (File Transfer)	C/S	C/S
TELNET (Terminal Access)		
VT100	C/S	C
VT220	C	C
3270	C	C
SMTP (Mail)	C/S	C
LPR/LPD (Print)	C/S	C/S
REXEC (Remote Command)	C/S	C
Network File System – KIT		
NFS Support	C/S	C
Application Toolkit		
Windows SOCKETS API	—	✓
SOCKET'S	✓	✓
RPC-SUN Remote Procedure Call	✓	✓
FTP API (File Transfer)	✓	✓
KERBEROS (Authentication)	✓	—
NCS (Network Computing)	✓	—
C=Client		S=Server

Ordering Information

Program No.		OTC
5871-AAA		
F/C 3437	IBM Personal Communications/3270 V3.0	\$495
5871-AAA		
F/C 2483	Networking Services/DOS	\$195
5875-XXX		
F/C 1595	IBM Multi-Terminal ASCII Emulator/2	\$300
5871-BBB	TCP/IP V2.1 for DOS	
F/C 3958	Base Kit	\$230
F/C 3959	NFS Kit	\$135
F/C 3961	Tool Kit	\$550
F/C 3960	NETBIOS	\$125
5871-BBB	TCP/IP V1.2 for OS/2	
F/C 2059	Base Kit	\$200
F/C 2061	NFS Kit	\$150
F/C 2064	Tool Kit	\$500
F/C 2062	X-Windows Kit	\$150
F/C 1716	NETBIOS Kit	\$176
F/C 2060	X.25 Kit	\$150

More Stuff.

IBM Print Services Facility/2

Advanced printing for your LAN.

Bring the power and sophistication of IBM's Advanced Function Printing™ technology to your LAN with IBM Print Services Facility/2™. Version 1.0 functions as a LAN print server, supporting multiple workstations, data streams and printer types — a great solution for your high-function, high-capacity or departmental printer requirements. For your distributed printing needs Version 1.10 is the best solution. It consolidates printing throughout the enterprise by allowing both the host and workstation application data to be printed on a wide range of LAN-attached printers. Version 1.0:

- Supports DOS, Windows, OS/2, UNIX and AIX.
- Supports all IBM IPDS Page Printers as well as other printer data streams, including HP PCL and PPDS.
- Provides local control of spools, printers and resources.
- Server runs OS/2.
- Includes host and LAN printer sharing solution.
- Supports S/370 and AS/400 via file transfer.

Additionally, Version 1.10:

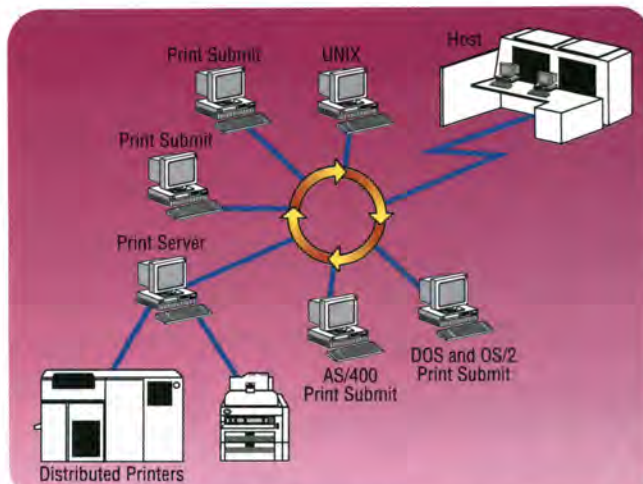
- Supports PostScript data stream and Adobe® Type 1 font.
- Works with IBM OS/2 and Novell NetWare® LANs.

RUMBA/400 for Microsoft Windows

Direct to your AS/400 files.

Install RUMBA/400™ on any personal computer running Microsoft Windows 3.0 and you have a graphical interface to access host information directly from a workstation. With RUMBA/400 you can create direct links that simultaneously update your PC files with the same changes you've made in the corresponding host application. Available as a feature of OS/400 PC support with V2R2 or as a standalone product for V2R1 or V2R1.1.

- Provides the tools to define your own text hot spots and activate common AS/400 or PC functions.
- Makes macro customization easy.
- EHLLAPI and DDE interfaces and Context Sensitive Help are available.
- Supports SAA OfficeVision™/400 and 5250 display and printer emulation.



PSF/2 gives you seamless S/370 and AS/400 host and LAN printing.

IBM Multimedia Presentation Manager/2

Presentations come to life!

MMPM/2 is the multimedia platform for today and tomorrow. It takes advantage of OS/2 2.0 features that make multimedia effective. This includes multitasking, which is essential for playing multiple data streams concurrently, and for playing synchronized audio and video.

MMPM/2 is also the multimedia platform for the future. Its extendable architecture enables new functions, devices and multimedia data types and formats to be added as the technology advances.

- Provides multimedia extensions to the OS/2 32-bit environment.
- MMPM Toolkit/2 contains C language bindings, sample programs and documentation to assist you in the use of MMPM/2.
- Recognizes a variety of audio and image data types.
- Provides open extendable architecture, data standards and consistent user interface.
- Provides synchronization support by taking advantage of the multitasking capabilities in OS/2 2.0.
- Provides CD-ROM/XA interleaved data and compressed audio support.

Trademarks used in this catalog: IBM, OS/2, OS/400, AS/400, DB2, The Integrated Reasoning Shell, APL2 and AIX are registered trademarks of International Business Machines Corporation. SAA, SQL/DS, Presentation Manager, RISC System/6000, CICS OS/2, BookManager, CUA, MVS, VM, Print Services Facility, Advanced Function Printing, S/390, OfficeVision and TIRS are trademarks of International Business Machines Corporation. UNIX is a registered trademark of UNIX System Laboratories, Inc. RUMBA/400, Windows and Word are trademarks and Microsoft is a registered trademark of Microsoft Corporation. WordPerfect is a registered trademark of WordPerfect Corporation. KnowledgeWare is a registered trademark of KnowledgeWare, Inc. InfoSpan and IRMS are trademarks of InfoSpan Corporation. BACHMAN/Database Administrator, BACHMAN/DBA, BACHMAN/Analyst, BACHMAN/Designer and

Put a book in your window.

Easily find and manage your company's increasing amounts of information with the IBM BookManager™ family of products — BUILD and READ. With the IBM SAA BookManager BUILD/2, you can convert documents created with WordPerfect® and Microsoft Word™ into online-readable, electronic books. These books can then be accessed and viewed using either the IBM SAA BookManager READ/2 or BookManager READ/DOS products. These programs organize books into electronic bookshelves allowing you to access information immediately through sophisticated search and link capabilities, within and across books.

Users' books and documents can be built and placed on the electronic bookshelf right next to softcopy books from IBM, and numerous other information providers who utilize the BookManager format.

- Get quick access to softcopy documents stored on a LAN server, AS/400 folder, diskette or CD-ROM.
- Able to use softcopy books in many languages.
- Displays image data and graphics within softcopy publications.
- Displays more than one book at a time with OS/2 Presentation Manager multitasking.
- BookManager READ/DOS 1.2 allows users to find, read and use information at their DOS workstations.

Ordering Information

Program No.	OTC
5871-AAA Print Services Facility/2	
F/C 2556 PSF/2 V1.0	\$2,900
F/C 2564 PSF/2 V1.1	\$2,900
5738-PC1 PC Support/400	
F/C 0795 RUMBA/400 Workstation Feature	\$365
5799-PLT RUMBA/400 PRPQ	\$414
BookManager Tools	
5601-454 BookManager READ/2 V1.2.1	\$203
5601-453 BookManager READ/DOS 1.2	\$203
5871-BBB*	
F/C 3779 BookManager BUILD/2 V1.2.1	\$895
5871-AAA Multimedia Presentation Manager	
F/C 2592 MMPM/2	\$125
F/C2593 MMPM Toolkit/2	\$199

*Limited availability scheduled 12/92. Call for details.

BACHMAN/Capture are trademarks of Bachman Information Systems, Inc. Micro Focus is a registered trademark and Micro Focus COBOL, Micro Focus CICS Option, Micro Focus CICS OS/2 Option, Micro Focus IMS Option and Micro Focus IMS Production Option are trademarks of Micro Focus Limited. Novell, NetWare and NetView are registered trademarks of Novell, Inc. i386 and i486 are trademarks of Intel Corporation. HP is a registered trademark of Hewlett-Packard Company. Apple is a registered trademark of Apple Computer Corporation. SUN is a registered trademark of SUN Microsystems, Inc. Adobe is a registered trademark of Adobe Systems Inc. DEC is a registered trademark of Digital Equipment Corporation. EASEL Workbench is a registered trademark of Easel Corporation.

Order Worksheet

To order, just call: 1 800 IBM-CALL

Fax to: 1 303 939-2014

Mail to:

IBM Corporation
Catalog Solutions Order Center
P.O. Box 2150
Department S18
Atlanta, GA 30301

Ship to:

Name

Title

Company

Address

City

State

Zip

Phone ()

Product	Program Number	Feature Code (if applicable)	Media (if applicable)	Price

Order direct from IBM.

For the convenience of ordering by phone, call 1 800 IBM-CALL from 8:00 a.m. to 7:00 p.m. ET, Monday - Friday.

Send no money now.

Send no money now. Payment is due at the end of the test period. If no test period applies, payment is due upon receipt of invoice. All orders are subject to normal IBM credit approval.

Free shipping!

We'll process your order immediately and you may expect delivery in 2-3 weeks. All domestic (continental U.S.) software shipments are made at no charge.

Licensing information for IBM programs.

The IBM programs featured in this catalog are offered under the terms of either the IBM Customer Agreement (ICA) or the IBM Program License Agreement (PLA). Program warranty information is included with the program or in the IBM Customer Agreement. Complete documentation regarding licensing terms, warranties, current prices, contracts, volume discounts and product descriptions are available by phone, fax or mail.

Cooperative Software Programs

Cooperative Software Programs **CSP** are provided by IBM Business Partners and marketed by IBM acting as an agent.

Prices in this catalog are subject to change without notice and do not include applicable sales tax. The symbol OTC refers to a One Time Charge for the product.

More. And more.

Power tools that give you more.

**IBM Personal
Application
System**

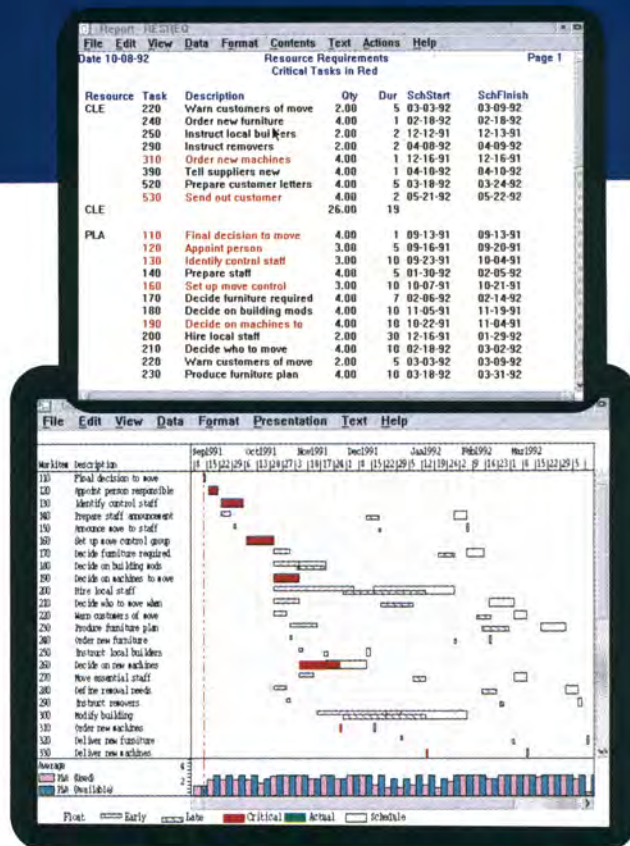
Project planning gets a boost.

IBM SAA Personal AS/2 is designed to work with IBM's Systems Application

Architecture environments. It provides OS/2 users with a combination of powerful facilities to gather, analyze and present the information needed to make better business decisions. With the help of the Project Management Module, developers can plan and schedule projects, monitor the progress of project activities and evaluate projects on completion.

Also available for Windows, Personal AS creates fully integrated customized solutions to your business problems. Both OS/2 and Windows versions:

- Enable you to access data in a wide range of personal computer data formats, OS/2 Relational, AS/400 and System/370 data via Application System.
- Give you comprehensive report and charting capabilities to create and customize reports and charts in the style you need.
- Visually link processes together in a procedure that can be run later against the current information.
- Allow you to select data, options and actions right on the screen with easy-to-use windowing feature.
- Optional packages for Statistics, Application Development, Business Planning and Project Management (OS/2 only) enhance your decision-making capabilities.



Ordering Information

Program No.		OTC
5871-BBB	Personal Application System	
F/C 0028	Personal AS/2 Base	\$763
F/C 0029	Development/2	\$848
F/C 0039	Statistics/2	\$485
F/C 0060	Business Planning/2	\$485
F/C 0062	Project Management/2	\$848
F/C 2530	Personal AS/DOS for Windows	\$763
F/C 2531	Development for Windows	\$848
F/C 2532	Statistics for Windows	\$485
F/C 2533	Business Planning for Windows	\$485

**Free
catalogs**

Choose the solution right for you.

From educational software programs to multimedia solutions, we've got the catalog to fit your computing needs. Simply dial 1 800 IBM-CALL and we'll gladly send you, free of charge, any of the catalogs listed here or help you find the catalog right for you.



© International Business Machines Corporation 1992

IBM Corporation
Catalog Solutions Order Center
Department S18 AB
P.O. Box 2150
Atlanta, GA 30301



**AS/400 Solutions
Catalog**

RISC System/6000 Direct Order Catalog



**Systems and Products
Guide**



handle is used for subsequent access to the device driver.

When an application makes a call to a device driver, the kernel intercepts the call and formats the device driver request into a standard request packet, which contains data and pointers that the device driver uses to complete the request. In the case of a `DosRead` or `DosWrite` command, the request packet contains the verified and locked physical address of the caller's buffer. In the case of an `IOctl`, the request packet contains the virtual address of a data and parameter buffer. Depending on the type of request, the data in the request packet changes but its header length and format remain fixed. The kernel sends the request packet to the driver by passing a 16:16 pointer to the packet.

OS/2 device drivers are loaded at boot time. The kernel keeps a linked list of installed device drivers by name, using the link pointer in the device header. Before a device driver is used, it must be `DosOpened` from the application. The `DosOpen` command specifies an ASCII-Z string with the device name as a parameter. The device name is an eight-character ASCII name located in the device header. The kernel compares this name with its list of installed device drivers and, if it finds a match, calls the `OPEN` section of the device driver strategy routine to open the device. If the device opens successfully, the kernel returns to the application a handle to use for future device driver access. The device handles are usually assigned sequentially, starting with 3, as 0, 1, and 2 are claimed by OS/2. However, handle value should never be assumed.

DEVICE DRIVER MODES

OS/2 2.0 device drivers operate in three different modes. The first, `INIT` mode, is a special mode entered during system boot and executed at Ring 3. When the OS/2 system loader encounters a `DEVICE=` statement in the `CONFIG.SYS` file, it loads the device driver `.SYS` file and calls the `Init` function of the device driver. What makes this mode special is that the boot procedure is running in Ring 3, which normally has no I/O privileges. In this mode, however, OS/2 allows a limited number of Ring 0 operations. The device

driver is free to do port I/O and turn off the interrupts, but it must insure that they are on before exiting the `INIT` routine. The `INIT` routine can be used to initialize a universal asynchronous receiver transmitter (UART) or anything else necessary to ready a device.

Ring 3 operation during `INIT` is necessary to protect the integrity of code that has already been loaded and to make sure that the device driver itself does not corrupt the operating system during initialization. During `INIT`, the device driver can call a limited number of system API routines to aid in the initialization process. For example, a device driver might use the API routines to read a disk file that contains data to initialize an adapter. The device driver also uses the API routines to display driver error and sign-on messages.

```
typedef struct ReqPacket {
    UCHAR   RPLength;      // request packet length
    UCHAR   RPunit;        // unit code for block DD only
    UCHAR   RPcommand;     // command code
    USHORT  RPstatus;      // status word
    UCHAR   RPreserved[4]; // reserved bytes
    UCHAR   RPqLink;       // queue linkage
    UCHAR   avail[19];     // command specific data
} REQPACKET;
```

Figure 2: OS/2 request packet

The second mode, called `Kernel mode`, is in effect when the device driver is called by the kernel as a result of an I/O request.

The third mode, called `Interrupt mode`, is in effect when the device driver's interrupt handler executes in response to an external interrupt such as a character being received from a serial port.

In general, the OS/2 device driver consists of a strategy section, an `INIT` section, and optional interrupt and timer sections. The strategy section receives requests from the kernel in the form of a request packet. The strategy section verifies the request, and if it can be completed immediately, completes the request and sends the result back to the kernel. If the request cannot be completed immediately, the device driver optionally queues the request to be completed at a later time and, if necessary,



starts the I/O operation. The kernel calls the strategy routine directly by finding its offset address in the device header.

REQUEST PACKETS

The first entry in the request packet is the request packet length, filled in by the kernel. The second parameter is the unit code.

```
typedef struct DeviceHdr {
    struct DeviceHdr far *DHnext; /* ptr to next header */
    USHORT DHattribute; /* device attribute word */
    OFF DHstrategy; /* offset of strategy routine */
    OFF DHidc; /* offset of IDC routine */
    UCHAR DHname[8]; /* dev name (char) or #units */
    char reserved[8];
} DEVICEHDR;

DEVICEHDR devhdr = {
    (void far *) 0xFFFFFFFF, /* link */
    (DAW_CHR | DAW_OPN | DAW_LEVEL1), /* attribute */
    (OFF) STRAT, /* &strategy */
    (OFF) 0, /* &IDCroutine */
    "DEVICE1 ", /* device name */
};
```

Figure 3: OS/2 device driver header

```
DEVICEHDR devhdr[2] = {
    { (void far *) &devhdr[1], /* link to next dev*/
      (DAW_CHR | DAW_OPN | DAW_LEVEL1), /* attribute */
      (OFF) STRAT1, /* &strategy */
      (OFF) 0, /* &IDCroutine */
      "DEVICE1 ",
    },
    {(void far *) 0xFFFFFFFF, /*link(no more devs)*/
      (DAW_CHR | DAW_OPN | DAW_LEVEL1), /* attribute */
      (OFF) STRAT2, /* &strategy */
      (OFF) 0, /* &IDCroutine */
      "DEVICE2 ",
    }
};
```

Figure 4: Device driver header, multiple devices

Applicable for block devices only, this field should be set by the device driver writer to zero for the first unit, one for the second, and so on. The third field, the command code, is filled in by the kernel. This is the code used by the switch statement in the strategy section to decode the type of request from the kernel. The next field is the status word returned to the

kernel. This field will contain the result of the device driver operation, along with the **DONE** bit, to notify the kernel that the request is complete. (This is not always the case; the device driver may return without the **DONE** bit set.) To make things easier, a C language union should be used to access specific types of requests. The request packet structures are placed in an include file, which is included by the device driver mainline.

THE DEVICE HEADER

A simple OS/2 device driver consists of at least one code segment and at least one data segment, although more memory can be allocated if necessary, as in Figure 2. The first set of data that appears in the data segment must be the device driver header, a fixed-length linked list structure that contains information for use by the kernel during **INIT** and normal operation, as in Figure 3.

The first entry in the header is a link pointer to the next device the device driver supports. If no other devices are supported, the pointer is set to - 1L. A - 1L terminates the list of devices supported by the driver. If the driver supports multiple devices, such as a four-port serial board or multiple disk controller, the link is a far pointer to the next device header, as shown in Figure 4. When OS/2 loads device drivers at **INIT** time, it forms a linked list of all driver device headers. The last device driver header will have a link address of - 1L. When a **DEVICE=** statement is found in **CONFIG.SYS**, the last loaded device driver's link pointer is set to point to the new driver's device header, and the new driver's link pointer now terminates the list.

The second entry in the device header is the device attribute word, which is used to define the operational characteristics of the device driver.

The third device header entry is a one word offset to the device driver strategy routine. Only the offset is necessary because the device driver is written in the small model with a 64K code segment and a 64K data segment. (This is not always true; in special cases, the device driver can allocate more code and data space if needed, and can even be written in the large model.)

The next entry in the device header is an offset address to an **IDC** routine, if the device driver

Arcadia Workplace Companion™

An **OS/2** Personal Information Manager



The Arcadia Workplace Companion (WPC) is a set of productivity tools designed specifically for the OS/2 Workplace Shell environment from the ground up! It adheres to Common User Access (CUA) '91 guidelines and completely integrates its various components in an object-oriented fashion. The WPC exploits all of the powerful new controls found within OS/2 2.0 including notebooks, containers, sliders, value sets, and spin buttons.

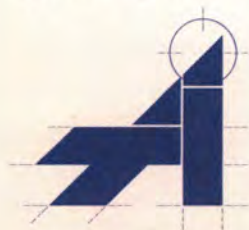
Everything necessary to schedule the day's events, keep track of appointments, and manage business contacts is represented graphically. The Workplace Companion contains a clock/calendar module, an appointment book, a telephone/address book (with phone log), a notepad, a to-do list, and other productivity features.

Finally, an
Intuitive personal
Information
management tool
for
OS/2 2.0!

Clock/Calendar. This module contains a month calendar, a full-year calendar, Julian date, analog/digital clock, international times, holidays, and more! It allows you to customize its many features, such as setting your own alarms and adding new cities to the international cities list.

Appointment Book. The Appointment Book is an excellent way to schedule your day's events. Appointments may be added by clicking on the time you wish to schedule and typing a description.

Telephone/Address Book. This module is a convenient way to store the name, numbers, and addresses of all your important contacts. The built-in Phone Log and Follow-up feature make tracking phone calls easy.



**Arcadia
Technologies
Inc.**

OS/2, CUA, IBM are registered trademarks of International Business Machines, Corporation. Arcadia Workplace Companion is a trademark of Arcadia Technologies, Incorporated.

735 West Duarte Road, Suite 207
Arcadia, CA 91007

Tel: (818) 446-6945
Fax: (818) 447-4212





supports interdevice driver communications. (The `DAW_IDC` bit in the device attribute word must also be set or the `AttachDD` call from the other device driver will fail.) The last field is the device name, which must be eight characters in length. Names with less than eight characters must be blank filled. Any mistake in coding the device driver header will cause an immediate crash and burn when booting.

DEVICE ATTRIBUTE WORD

Bits	Description
15	Set if character driver, 0 if block driver
14	Set if driver supports interdevice communications (IDC)
13	For block drivers, set if non-IBM format; for character drivers, set if driver supports output-until-busy
12	If set, device supports sharing
11	Set, if block device supports removable media; if character device supports device open and close
10	Reserved, must be 0
9-7	Driver function level; 001 = OS/2 device driver 010 = supports <code>DosDevIOctl2</code> 011 = capabilities bit strip in device header
6-4	Reserved, must be 0
3	Set if this is the <code>CLOCK</code> device
2	Set if this is a null device (character driver only)
1	Set if this is the new <code>stdout</code> device
0	Set if this is the new <code>stdin</code> device

Table 1: Device attribute word

CAPABILITIES BIT STRIP

Bits	Description
0	Set if driver supports <code>DosDevIOctl2</code> packets and has shutdown support
1	For character drivers, set if driver supports 32-bit memory addressing; for block drivers this bit must be 0
2	If set, the device driver supports parallel ports
3-31	Reserved, must be 0

Table 2: Capabilities bit strip

Device Attribute Word

The device attribute word describes the type of driver and its capabilities. OS/2 uses this information for calling the device driver. Device attribute word bits are described in Table 1.

Capabilities Bit Strip

The capabilities bit strip word defines additional features supported on level 3 drivers only, as shown in Table 2.

THE STRATEGY SECTION

The strategy section is nothing more than a big switch statement, as shown in Figure 5. Common device driver requests, such as `DosWrite` and `DosRead`, have predefined function codes assigned to them. The device driver may elect to ignore any or all of these requests by returning a `DONE` status to the kernel, noting that the request has been completed. The status returned to the kernel may include error information, which the kernel returns to the calling program.

In the case of one of the standard device driver functions, the kernel maps the error value returned from the device driver to one of the standard device driver return codes. It is therefore impossible to pass any special return codes to the application by way of a standard device driver request.

If the device driver must return special error codes, it must use an `IOctl` request. `IOctls` are used for special types of device driver-specific operations such as port I/O that don't fit into the architecture of the standard device driver functions. The `IOctl` section of the device driver is called when the application issues a `DosDevIOctl` call with the device driver's handle. Using `IOctls`, the device driver can return specialized codes that might contain, for example, the contents of an I/O port or the status of the device. This flexibility allows the driver writer to customize the device driver to fit any device.

The strategy section is entered when the kernel calls the device driver to perform a particular operation.



CELEBRATE THE POWER

AM(Applications Manager)...the only OS/2 client/server development tool providing 32-bit applications exploiting the Workplace Shell and accessing data from local LAN or remote host databases. **AM** increases productivity at least 10-fold compared to conventional application development languages. **AM** is currently in use by over 1,000 organizations worldwide and is the leader in the delivery of proven, live applications.

You too can experience the power...**CALL NOW...508-640-1080.**

INTELLIGENT ENVIRONMENTS INC. 2 HIGHWOOD DRIVE, TOWNSBURY, MA. ALL PRODUCTS AND PRODUCT NAMES MENTIONED ARE REGISTERED TRADEMARKS OF THEIR RESPECTIVE COMPANIES.




INITIALIZATION

The INIT call is made only once during system loading in response to a `DEVICE=` in the `CONFIG.SYS` file. The call is made in the `INIT` mode from Ring 3, but with I/O privileges. The `INIT` routine is generally used at a time when code is inserted to initialize a device, such as setting up a serial port or verifying that the correct hardware is installed.

```
int main(PREPACKET rp, int dev)
{
    switch(rp->RPcommand)
    {
        case RPINIT:          /* 0x00 */

            /* init called by kernel in protected mode */
            return Init(rp);

        case RPREAD:          /* 0x04 */
            return (RPDONE);

        case RPWRITE:         /* 0x08 */
            return (RPDONE);

        case RPINPUT_FLUSH:   /* 0x07 */
            return (RPDONE);

        case RPOUTPUT_FLUSH:  /* 0x0b */
            return (RPDONE);

        case RPOPEN:          /* 0x0d */
            return (RPDONE);

        case RPCLOSE:         /* 0x0e */
            return (RPDONE);

        case RPIOCTL:         /* 0x10 */
            switch (rp->s.IOctl.function)
            {
                case 0x00:     /* our function def #1 */
                    return (RPDONE);

                case 0x01:     /* our function def #2 */
                    return (RPDONE);
            }

        /* deinstall request */
        case RPDEINSTALL:     /* 0x14 */
            return(RPDONE | RPERR | ERROR_BAD_COMMAND);

        /* all other commands are flagged */
        default:
            return(RPDONE | RPERR | ERROR_BAD_COMMAND);
    }
}
```

The first thing the driver should do during initialization is to save the `DevHlp` entry point address passed in the request packet. This is the only time that the address is made available to the device driver, and it must be saved in the device driver's data segment.

The `INIT` code generally performs two other functions. First, it issues a sign-on message to the screen that the device driver is attempting to load. Second, it finds the address of the last data and code item and sends them back to OS/2. OS/2 uses the code and data offset values to size memory. Only the first code and data segment of the device driver are resized by OS/2, so it may be desirable to place the `INIT` code and data into another segment, which is discarded after the device driver is loaded. If a device driver fails installation, it must send back zero offsets for its code and data segments so OS/2 can use the memory space the device driver occupied during installation.

If the driver supports multiple devices, it will contain a device header with an entry for each device, with one device header pointing to the next, as shown in Figure 6. The last device header contains a `-1L`, which terminates the list. For each device, the OS/2 kernel calls the strategy entry point to initialize the device. If the driver supports, for example, four serial ports that use a single interrupt level, only the last valid initialized device should hook the interrupt. This prevents previously installed devices from generating interrupts before initialization is completed. The code and data segment values returned to OS/2 to size memory should be exactly the same each time the `Init` section is called.

During `INIT`, a limited number of API functions can be called by the device driver. This is possible because `INIT` runs as a Ring 3 thread. These calls are especially helpful for initializing device drivers using data that is resident in files.

The driver should allocate necessary resources such as memory and GDT selectors during initialization. If the driver supports a memory mapped adapter, the physical adapter address can be mapped to a GDT selector. However, because `INIT` is performed as a Ring 3 thread, the GDT selector cannot be accessed during initialization.



If the driver is installed on an IBM PS/2™ Micro Channel™ machine, the device driver should search the system for an adapter card with the correct ID and verify that it is configured correctly. The device driver may also call special PS/2 Advanced BIOS (ABIOS) routines to verify the correct configuration.

A COMMON STRATEGY

One of the most common techniques in OS/2 device driver design is for the strategy section to request service from the device and wait for a device or timer interrupt to signal completion of the request. In this case, the strategy section starts the I/O and issues a `BLOCK DevHlp` call, which blocks the calling thread. When the device interrupt signals that the operation is done, the interrupt section runs the blocked thread, completing the request. To protect against the request never being completed, as with a down device, the `BLOCK` call can contain a time-out parameter. If time expires before the completion interrupt occurs, the `BLOCKED` thread will be run, allowing the strategy section to send the proper error message back to the kernel.

Another way to time-out a device is to use the `SetTimer DevHlp` routine. A timer handler can be hooked into the OS/2 system clock, and ticks counted until the timer handler decides to run the `BLOCKED` thread.

The number and type of commands supported by the strategy section are up to the designer of the device driver. The device driver can process only the commands it needs to, and let the others pass through, by sending a `DONE` status back to the kernel. Illegal function calls may, optionally, be trapped, and an `ERROR_BAD_COMMAND` returned to the kernel.

The OS/2 kernel periodically issues special requests to the device driver, such as the 5-48 Code Page IOCTL, which the kernel sends to every OS/2 device driver immediately following the `Open`:

If the application has a pending request blocked in the device driver when it failed, the kernel unblocks the device driver with an "unusual wakeup" return code. The device driver must then return `ERROR_CHAR_CALL_INTERRUPTED` to the kernel, which, in turn, calls the `CLOSE` section of the driver.

In general, it's a good practice to trap all unsupported requests by returning the `DONE` and `ERROR_BAD_COMMAND` status to the kernel.

In the simplest of device drivers, the strategy section may contain only `Open`, `Close`, and `Read` or `Write`. In a complicated driver such as a disk device driver, the strategy section may contain over two dozen standard device driver functions and dozens of additional IOCTL calls. IOCTL calls are actually strategy functions, but are broken down one step further to provide more detailed

```

DEVICEHDR devhdr [2] = {
    { (void far *) &devhdr [1],           //Link to next dev
      (DAW_CHR | DAW_OPN | DAW_LEVEL1),    //attribute
      (OFF) STRAT1,                        //&strategy
      (OFF) 0,                             //&IDCroutine
      "DEVICE1 ",
    },

    ((void far *) 0xFFFFFFFF,             //Link(no more devs)
      (DAW_CHR | DAW_OPN | DAW_LEVEL1),    //attribute
      (OFF) STRAT2,                        //&strategy
      (OFF) 0,                             //&IDCroutine
      "DEVICE2 ",
    },
};

```

Figure 6: Device driver header, multiple devices

or device-specific operations. For instance, an application might send a list of parameters to a device driver to be used in initializing an I/O port. This operation would not be possible using one of the standard device driver function calls because it is so device-specific. The `IDCTL`, however, is well suited to this type of function.

INTERRUPT SECTION

The interrupt section handles interrupts from the device. Interrupts may be caused by a character having been received, a character that has finished transmitting, or any number of external events. Interrupt processing should be quick and straightforward. The routine that handles the interrupt is appropriately called the interrupt handler, a subroutine entered at every interrupt for the `IRQ` level registered with the `SetIRQ DevHlp` call. All interrupts in OS/2 are handled by the kernel, a change from DOS, where a program had only to hook the interrupt vector it wanted. OS/2 does not allow interrupt vectors to be



changed, and if an attempt is made to change one, the application is immediately kicked off the system. To register for an OS/2 interrupt, the device driver must send the address of its interrupt handler and the requested interrupt (IRQ) level to OS/2 through a `SetIRQ DevHlp` call. If the `SetIRQ` is successful, OS/2 calls the interrupt handler upon receipt of an interrupt on that IRQ.

OS/2 calls each registered interrupt handler until one claims ownership of the interrupt by clearing the carry flag (CLC). The interrupt handler must be in the first code segment of the device driver.

OS/2 calls the device driver's interrupt handler with interrupts disabled if the device driver registered for the interrupt as nonshared (ISA bus). If the `SetIRQ` call is made with the shared interrupt option (Micro Channel, EISA), interrupts remain enabled when the interrupt handler is entered. Shared interrupts are a feature of the IBM Micro Channel and EISA bus architectures, which allow more than one device to share a single interrupt level.

Extended time in the interrupt handler can cause performance problems. Since the interrupt routine is frequently entered as a result of data being received, the interrupt handler must quickly store the data and exit. In the case of character devices, the OS/2 `DevHlp` library supports fast reads and writes to circular character queues. For block devices, interrupt handling is fast because the interrupt is usually caused by a DMA completion or disk-seek complete. For block devices, data is usually transferred to the user buffer using DMA, eliminating the need to transfer data during interrupt processing. On a DMA transfer, the DMA controller is set up and started, and the device driver exited to allow other threads to run. When the DMA completes, it generates a DMA completion interrupt, causing the device driver interrupt handler to be entered. The interrupt handler then takes the appropriate action, for example, starting a new DMA transfer.

Most UARTs and adapters have some type of buffering, which allows a device driver some slack in servicing higher data rates. When an interrupt occurs, the UART is examined to determine the type of interrupt: transmit, receive, or clock.

The interrupt handler is not entered directly from OS/2, but is called from a small assembly

driver startup language routine. When the `SetIRQ` call is made to register the interrupt handler, the address passed in the call is the address of the interrupt handler entry point in the device driver start-up code. The start-up code, in turn, calls the C interrupt handler.

The interrupt handler routine is not difficult to write or understand. It can, however, be difficult to debug. Errors that occur in the interrupt handler frequently appear only in a real-time context, that is, while the interrupt handler is being entered because of a hardware interrupt. The C library function `printf`, for example, cannot be called from within an interrupt handler. Application debuggers such as CodeView™ cannot be used in an interrupt handler; rather, a debugger such as the OS/2 KDB must be used. A breakpoint placed in the interrupt routine will cause the program to stop, and further interrupts may pass undetected while the program is stopped. A problem may not appear when breakpoints are inserted, but will reappear when the program executes normally. It then becomes necessary to visualize the operation of the interrupt handler and begin applying solutions until the problem is fixed.

The interrupt handler may receive unsolicited or spurious interrupts from the hardware, which should be handled. In the sample interrupt handler, a check is made to see whether a valid read or write request is pending. If not, the device is reset and the interrupt handler is exited, effectively ignoring the interrupt.

Steven J. Mastrianni, *Personal Systems Software Inc.*, 15 Great Oak Lane, Unionville, Conn., 06085 (203) 693-0404. Mastrianni is an independent consultant specializing in OS/2 2.0 device drivers and applications software. He has over 20 years' experience writing software for real-time systems. Mastrianni is the author of *Writing OS/2 2.0 Device Drivers in C* (Van Nostrand Reinhold, 1992).

REFERENCES

This article is excerpted from *Writing OS/2 Device Drivers in C*, by Steven Mastrianni (Van Nostrand Reinhold, 1992), IBM Doc. G362-0006. It can be ordered through the IBM Mechanicsburg distribution center or from the publisher by calling (800) 842-3636.



LAN Interoperability with OS/2 2.0

by Bob Spratt

In the past few months, a new buzzword has appeared in the area of computer networking. "Interoperability" is suddenly getting attention in the local area network (LAN) world.

Interoperability is commonly known as the ability to communicate with multiple communication protocols from a single system at a single instance in time. This definition can also be expanded to include the ability of a workstation to use the multiple protocols on a single network adapter without conflict. A single workstation can access different types of server resources on the same network at the same time with one or more network adapters. For example, an OS/2 workstation can share files simultaneously on a NetWare™ server, an OS/2 LAN server, a Vines™ server, and a TCP/IP NFS server on a single LAN.

This article describes interoperability and explains why it is necessary in today's LAN environment. It also shows how OS/2 2.0 provides the mechanisms necessary for easy interoperability and explains how to get an interoperable system working with programs available from IBM and other suppliers.

At the time this article was written, neither the NetWare nor the Vines requester was finalized, so there may be some minor discrepancies between this article and the actual final product, most likely in the section that describes how to install individual pieces. This article can still serve as a guide for creating an OS/2 requester that can communicate with multiple servers.

This article describes how to make interoperability work using token-ring (T/R) as an example. T/R is used for illustration purposes; it can be replaced with another supported network medium such as Ethernet.

INTEROPERABILITY

Developers may wonder about the practical applications of interoperability. Today's LAN environments use many different communication protocols, ranging from NetBios and IEEE 802.2 in the OS/2 LAN Server environment to IPX and SPX in the NetWare environment. There are many other protocols, including Vines IP from Banyan and TCP/IP. In most large corporate networks and many smaller networks, these protocols are being used to communicate between different types of machines. Interoperability gives users the ability to access resources on any machine on the network, regardless of the protocol being spoken.

The network in Figure 1, for example, consists of three PS/2 Model 95s running NetWare

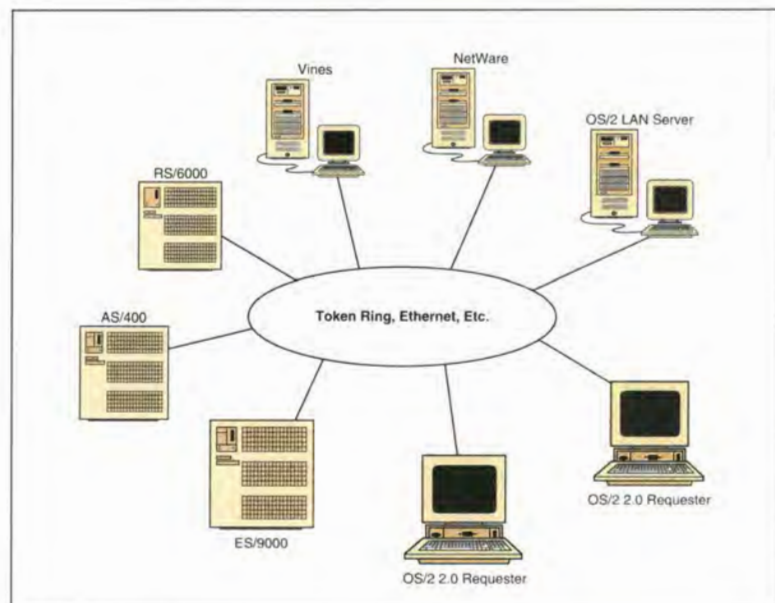


Figure 1: Network example

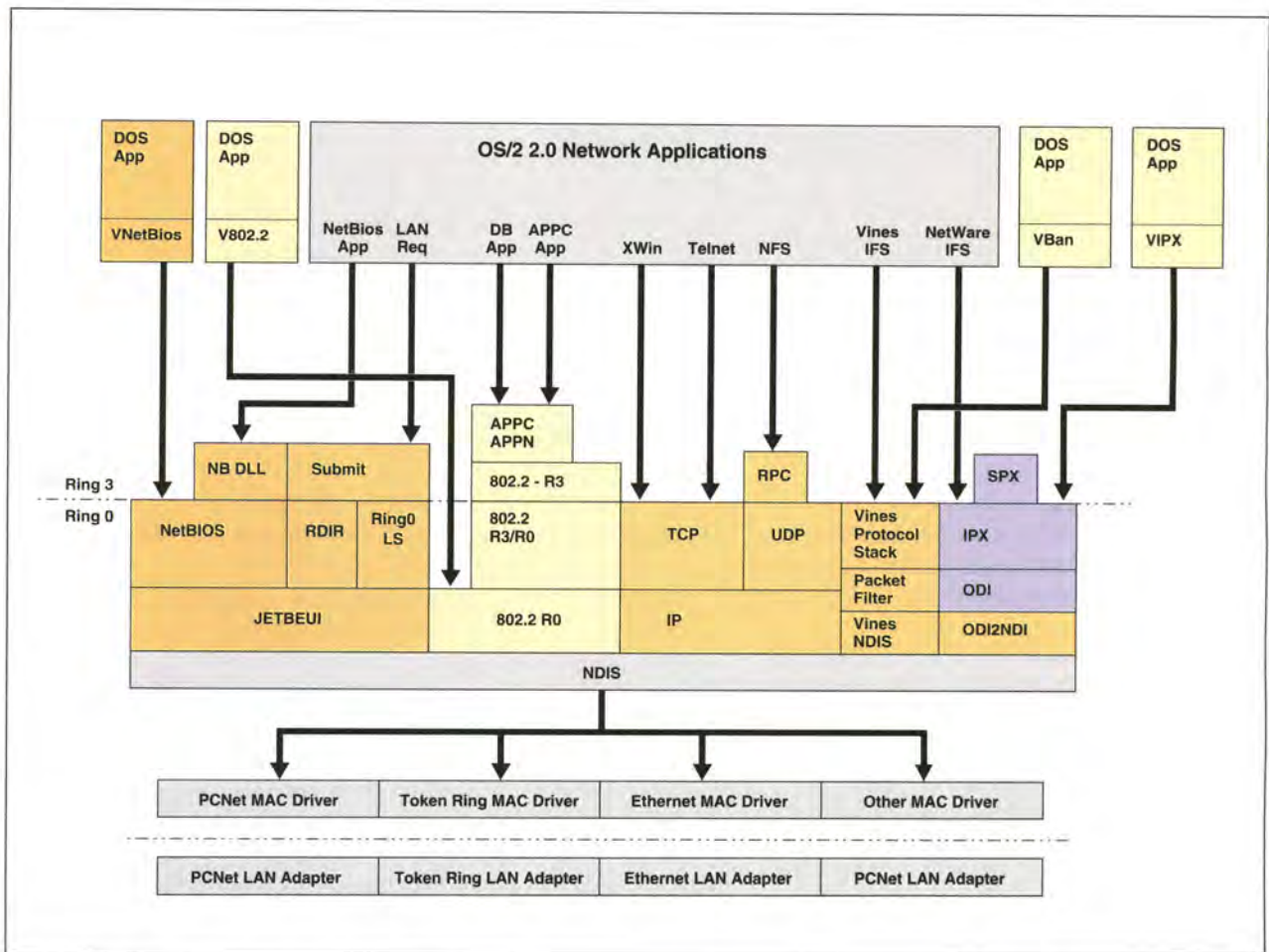


Figure 2: The NDIS protocol stack with ODI support

from IBM, OS/2 LAN Server, and Banyan Vines. Other types of computers are also present on the LAN: an RS/6000, an AS/400, and an ES/9000. Each of these machines provides some resource to the user at a workstation. With interoperability, a user can access resources on any machine.

OS/2 provides the perfect base operating system for this situation, one with connective capabilities that allow multiple communications protocols to be spoken over a single network adapter. This eliminates the need for multiple adapters to support multiple protocols, as was the case with early operating systems.

Interoperability is not limited to LANs, but can be extended to include other communications protocols. OS/2 Extended Services (ES) includes Communications Manager, a product that can communicate with host systems such as the ES/9000. System links using ES and LS

are maintained when used in conjunction with other protocols such as Vines and NetWare.

All these protocols have implementations specifically designed for OS/2 2.0, and all except NetWare are based on the Network Driver Interface Specification, or NDIS. (NetWare is based on the ODI specification; as a special case, it will be discussed later.)

WHAT IS NDIS?

NDIS, jointly developed by Microsoft and 3Com, provides a network driver architecture and interface that allows a system to support multiple network adapters and protocols. It provides a mechanism for managing these protocols and adapters so that different protocols can be used on different cards.

As a working definition, NDIS is a mechanism to support one or more network protocols on

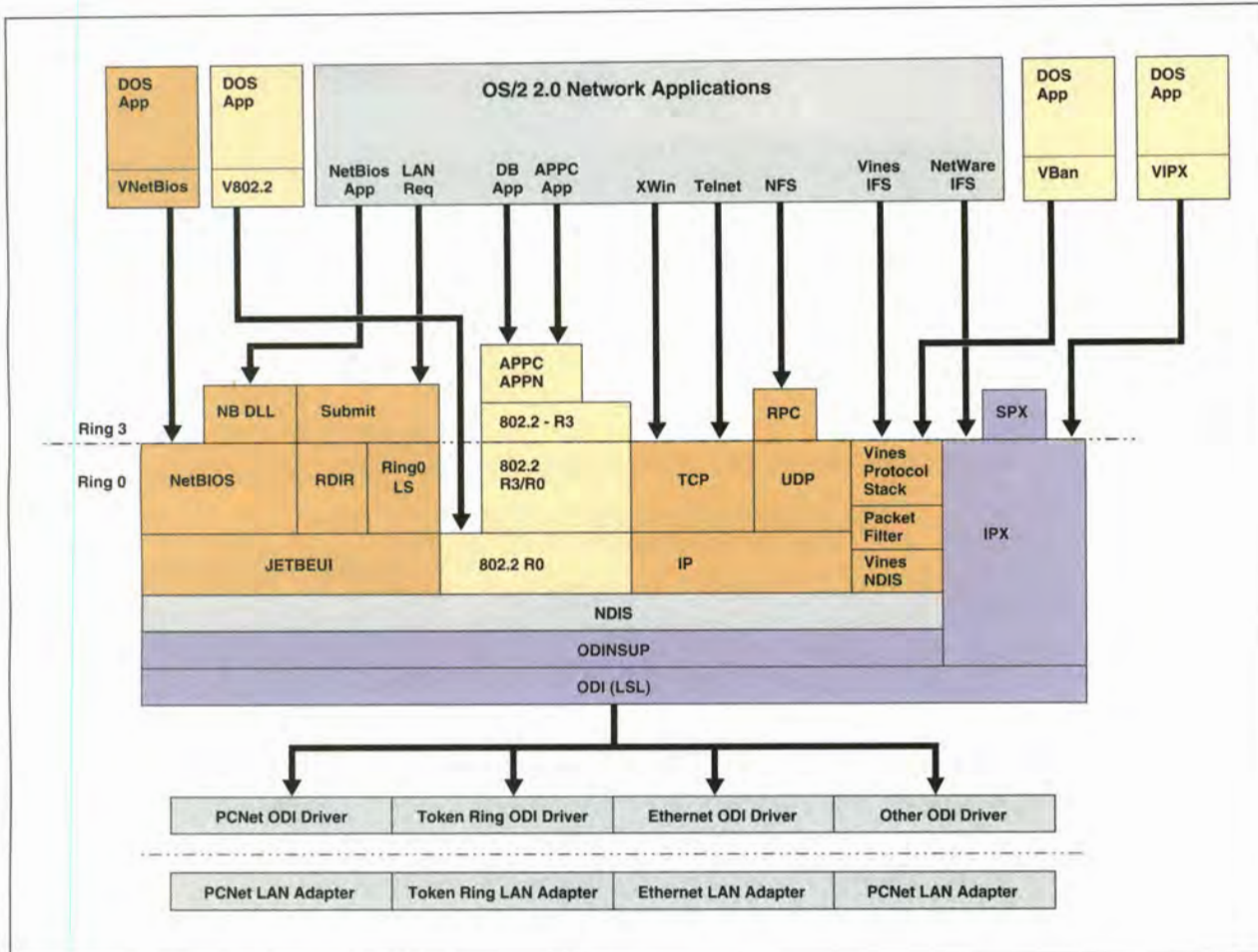


Figure 3: The ODI protocol stack with NDIS support

one or more network adapters. The adapters may or may not be of the same type. For instance, Ethernet and T/R adapters can be used in the same system.

NDIS defines four types of drivers used to support communications.

Media Access Control (MAC) Driver

MAC drivers are traditional device drivers for the network adapter. They provide low-level access to the hardware. MAC drivers' primary responsibility is to provide basic packet transmission and reception functions.

Protocol Driver

Protocol drivers provide a higher level of support ranging from the data link to the application layers of the OSI model. Protocol drivers include NetBios and IEEE 802.2.

MAC-Layer Entities

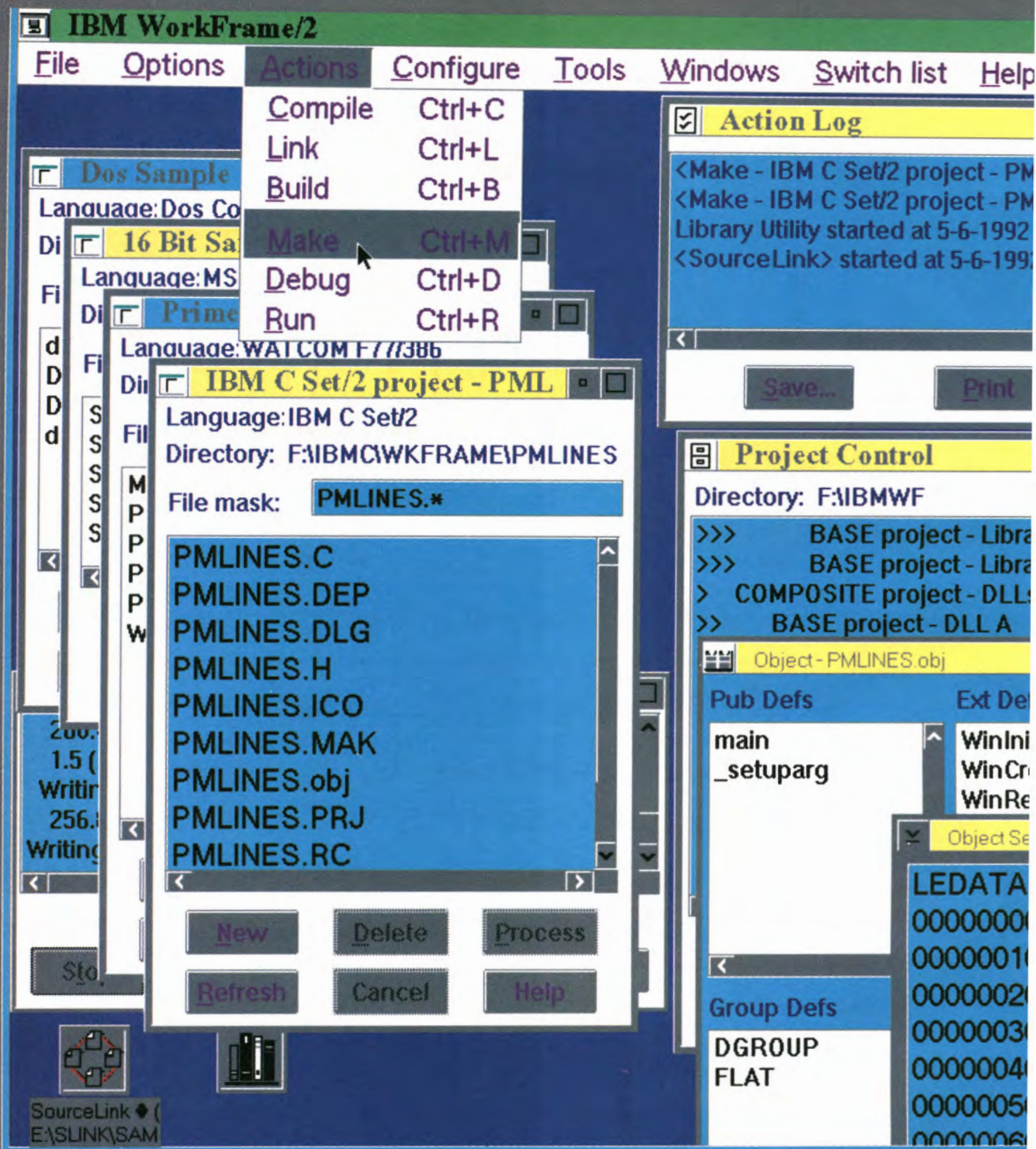
MAC-layer entities bring the MAC-layer interface up to a user level for tools and test purposes. Further discussion of these entities is beyond the scope of this article.

Protocol-Manager Driver

The protocol-manager driver provides a standard way for MAC and protocol drivers to receive configuration information about themselves and each other. It allows multiple drivers to bind together to create a desired protocol structure. Information used by the protocol manager is stored in the file `PROTOCOL.INI`.

WHAT IS ODI?

Novell defines ODI as open data-link interface. ODI supports media and protocol-independent communications with a standard



OS/2

To a software developer, this is what heaven looks like.

Most people wouldn't know what to make of a screen like this. But developers like you know a screen like this can help make all kinds of applications. With OS/2[®] 2.0, you can develop the DOS, Windows,[™] OS/2 and host-based apps end users need. And you can do it faster and easier than ever before. Because OS/2 2.0 can make the most of your 386 or 486 processor.

Now you can edit in one window, compile in another, profile in a third and test in a fourth. Pre-emptive multitasking makes everything run smoothly and responsively. And OS/2 Crash Protection[™] helps shield running applications from each other, so if one goes down it won't affect the others. Instead of rebooting, you just restart the affected app and continue. And since OS/2 2.0 is a 32-bit operating system, programs are easier to write and run faster, too. Which all adds up to improved productivity and reduced development cycle time.

But maybe the best part is that for less than the cost of DOS and Windows, OS/2 gives you a whole lot more. And to keep your cycle rolling, a full range of services and support are available, like on-line help through OS/2 Support line, Bulletin Board and IBM Link. Or you can join the IBMOS2 and OS2DEV conferences on CompuServe[®], where you can meet IBMers, users and developers who can find fast answers to your questions. For an IBM authorized dealer near you, or to order OS/2 2.0 from IBM, call 1 800 3-IBM-OS2.*

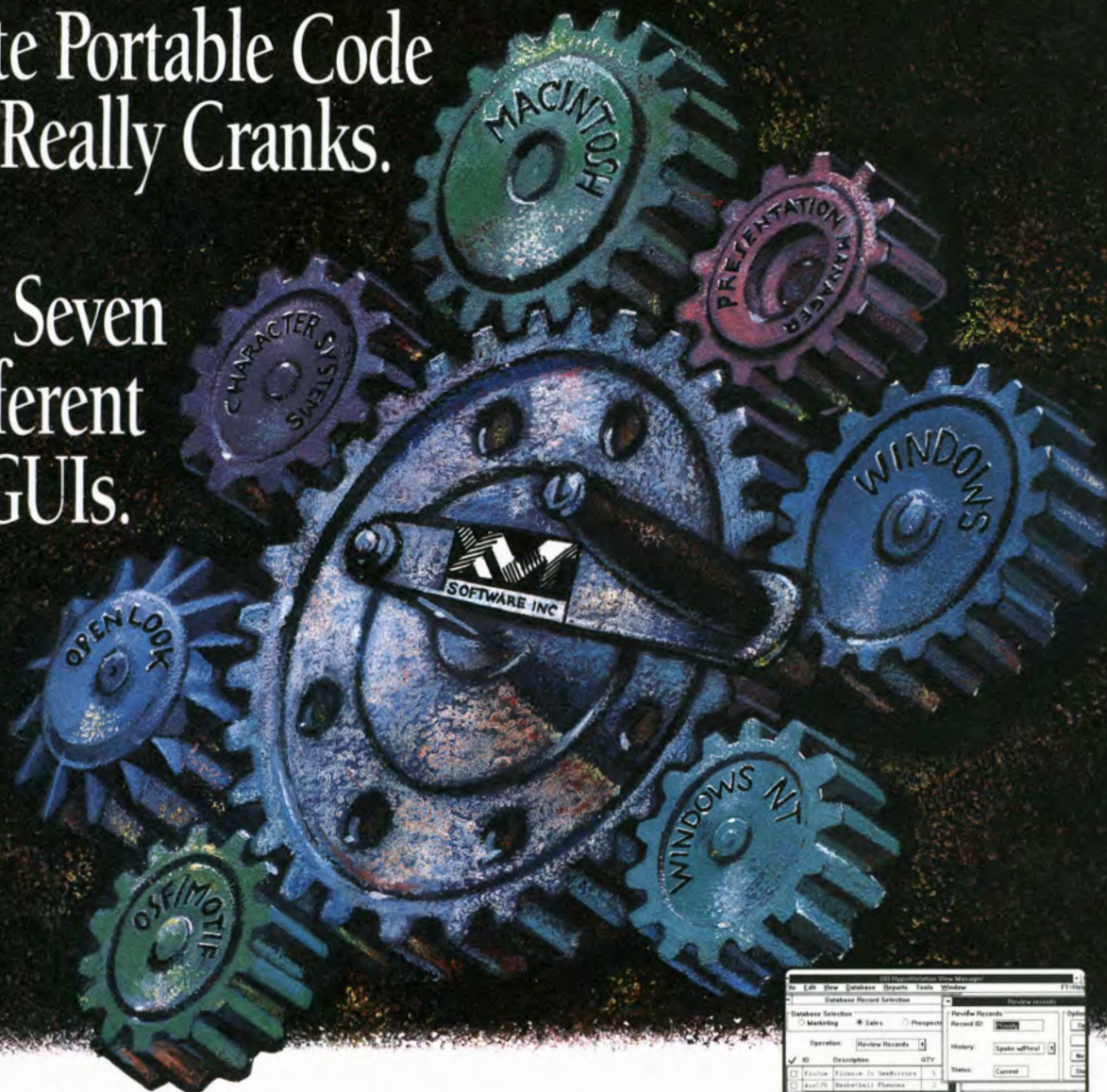


- OS/2 Crash Protection helps shield applications from each other.
- The integrating platform of choice for DOS, Windows and OS/2.
- Preemptive multitasking for responsive, reliable execution.
- 32-bit flat address space for productive programming.
- A full range of IBM services and support.



Write Portable Code That Really Cranks.

On Seven Different GUIs.



XVT's Portability Toolkit™ is a powerful C development environment that allows you to build a single application, then re-compile to every major GUI without rewriting code. XVT solutions also include an interactive design tool and a class library for C++ developers.

- Supports Macintosh, Microsoft Windows, Windows NT, OS/2 Presentation Manager, OPEN LOOK, OSF/Motif, and Character Systems
- Native look-and-feel to all target GUIs
- Portability to 26 hardware systems
- Access to the complete functionality of every windowing system
- Easier to use than native development toolkits
- Minimal size and performance overhead
- Shorter development cycles
- No royalties or runtime fees
- Clear documentation and responsive technical support

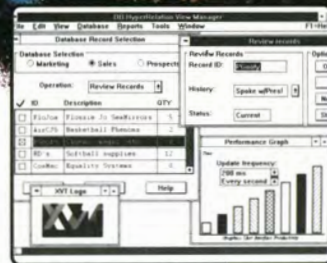
Now in its third generation, XVT is recognized as the industry leader in portable GUI development solutions and is the base document for the emerging IEEE standard. It is used by world-class software developers like •Novell •HP •AT&T •Digital •Lockheed •Kodak •Grammatik/Reference Software, because it allows them to take their applications to the widest market, quickly and cost effectively.

Don't write another line of code without gearing up to develop your application simultaneously for all GUIs. Call for technical materials and a demo.

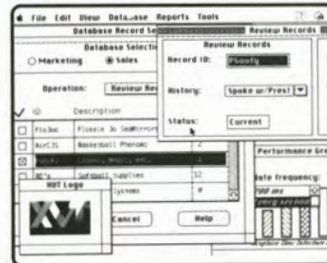


The portable GUI development solution.
1-800-678-7988

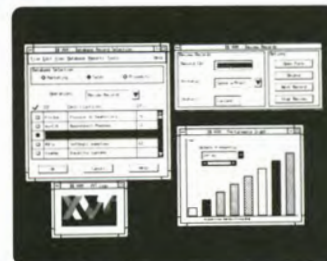
XVT Software Inc. 4900 Pearl East Cir. Boulder, CO 80301
(303) 443-4223 FAX (303) 443-0969
For European inquiries, contact: PVI Precision Software GmbH
Phone: 49 0 61 03/37 94 0 Fax: 49 0 61 03/36 95 5



▲ Microsoft Windows & Windows NT



▲ Macintosh



▲ OSF / Motif

Shown above are four of the seven GUIs supported by XVT.

XVT and XVT Portability Toolkit are trademarks of XVT Software Inc. Other product and company names are trademarks or registered trademarks of their respective owners.



```

DEVICE=C:\IBMCOM\LANMSGDD.OS2 /I:C:\IBMCOM
DEVICE=C:\ibmcom\protman.os2 /I:C:\ibmcom
DEVICE=C:\ibmcom\protocol\LANDD.OS2
DEVICE=C:\ibmcom\protocol\LANDLDD.OS2

LIBPATH=C:\MUGLIB\DLL;C:\IBMLAN\NETLIB;C:\NETWARE;C:\VINES\DLL;C:\TCPIP\DLL;C:\ibmcom\dll;
SET PATH=C:\IBMLAN\NETPROG;C:\MUGLIB;L:\OS2;P:\OS2;C:\NETWARE;Z:\;C:\TCPIP\BIN;
SET DPATH=C:\IBMLAN\NETPROG;C:\IBMLAN;C:\MUGLIB\DLL;C:\NETWARE;z\;C:\ibmcom;
SET HELP=C:\TCPIP\HELP;
DEVICE=C:\IBMCOM\PROTOCOL\LANPDD.OS2
DEVICE=C:\IBMCOM\PROTOCOL\LANVDD.OS2

SET BOOKSHELF=C:\IBMLAN\BOOK;

REM Comment out the following line if the Novell solution to interoperability
REM is being used
DEVICE=C:\IBMCOM\MACS\IBMTOK.OS2

REM -- NETWARE REQUESTER STATEMENTS BEGIN --
DEVICE=C:\NETWARE\LSL.SYS
RUN=C:\NETWARE\DDAEMON.EXE

REM Comment out the following line if the Novell solution to interoperability
REM is being used
device=C:\ibmcom\protocol\odi2ndi.os2

```

Figure 4: Example CONFIG.SYS (continued on page 32)

interface that allows transport protocols to share a single network board without conflict. The Novell equivalent to NDIS, ODI supports constructs similar to those in the NDIS model.

Multiple Link Interface Drivers (MLIDS)—

MLIDs correspond to the NDIS MAC driver, providing the same type of functions.

Protocol Stack—The protocol stack corresponds to the NDIS Protocol Driver. Examples include IPX/SPX and AppleTalk.

Link Support Layer (LSL)—The LSL roughly corresponds to the NDIS Protocol Manager. It is responsible for routing protocols to different adapters.

The most difficult aspect of interoperability with OS/2 2.0 is having ODI protocols and NDIS protocols exist concurrently. Both Novell and IBM have offered solutions. Figures 4 and 5 give IBM's solution, with changes needed for Novell's solution noted in the code.

IBM's solution uses its recently announced Network Adapter and Protocol Support (NAPS). Its code contains an additional protocol driver, ODI2NDI, that provides an interface between ODI layers and NDIS, allowing ODI-based protocols to communicate over the NDIS layer. Because all other OS/2 protocols are NDIS-based, this is the simplest solution. Figure 2 shows a diagram of protocol stacks supported by NDIS.

Novell's solution is similar to that of IBM, but puts an interface between NDIOS and ODI, allowing NDIS protocols to communicate over the ODI interface layer. Novell adds an additional protocol driver called ODINSUP, which adds layers to all protocol stacks in the system except NetWare. Figure 3 shows how ODINSUP connects the NDIS protocols to ODI; comparing it to Figure 2 shows the differences between the two approaches and illustrates path length differences between the upper layer applications and the bottom layer adapter device drivers.



```

REM Add the following line if the Novell solution to interoperability
REM is being used
REM DEVICE=C:\NETWARE\TOKEN.SYS

DEVICE=C:\NETWARE\ROUTE.SYS
DEVICE=C:\NETWARE\IPX.SYS
DEVICE=C:\NETWARE\SPX.SYS
RUN=C:\NETWARE\SPDAEMON.EXE
REM DEVICE=C:\NETWARE\NMPIPE.SYS
REM DEVICE=C:\NETWARE\NPSEVER.SYS
REM RUN=C:\NETWARE\NPDAEMON.EXE NP_COMPUTERNAME
DEVICE=C:\NETWARE\NWREQ.SYS
IFS=C:\NETWARE\NWIFS.IFS
RUN=C:\NETWARE\NWDAMON.EXE

REM Do not use the NetBIOS support that comes with NetWare
REM DEVICE=C:\NETWARE\NETBIOS.SYS
REM RUN=C:\NETWARE\NBDAEMON.EXE

DEVICE=C:\NETWARE\VIPX.SYS
REM DEVICE=C:\NETWARE\VSHELL.SYS
REM -- NETWARE REQUESTER STATEMENTS END --

DEVICE=C:\IBMCOM\PROTOCOL\NETBEUI.OS2
DEVICE=C:\IBMLAN\NETPROG\RDRHELP.200
IFS=C:\IBMLAN\NETPROG\NETWKSTA.200 /I:C:\IBMLAN /N
DEVICE=C:\IBMCOM\PROTOCOL\NETBIOS.OS2

REM >THE NEXT FEW LINES LOAD VINES NETWORK SOFTWARE IN
REM >REQUIRED ORDER, SET AT INSTALLATION TIME. DON'T EDIT
REM >OR MOVE ANY LINE WITHOUT SPECIFIC VINES INSTRUCTION.
REM >ALSO, DON'T REMOVE "Z:\" FROM YOUR SET PATH=LINE
REM >OR "C:\vines-directory\DLL" FROM YOUR LIBPATH=LINE.
SET VINESDIR=C:\VINES
IFS=C:\VINES\VINES2.IFS
DEVICE=C:\VINES\BANCOMM2.SYS /SOCKETS=60 /SPP_CONNECTIONS=60 /KBYTES_COMMBUFFERS=96
DEVICE=C:\VINES\VBAN.SYS
DEVICE=C:\VINES\drivers\bn_NDIS\NDISBAN2.SYS
REM >END OF VINES SOFTWARE INFORMATION.

RUN=C:\IBMLAN\NETPROG\LSDAEMON.EXE

SET ETC=C:\TCPIP\ETC
SET TMP=C:\TCPIP\TMP
RUN=C:\TCPIP\BIN\CNTRL.EXE
IFS=C:\TCPIP\BIN\NFS200.IFS
DEVICE=C:\TCPIP\BIN\INET.SYS
DEVICE=C:\TCPIP\BIN\IFNDIS.SYS

RUN=C:\ibmcom\protocol\landll.exe
RUN=C:\ibmcom\protocol\netbind.exe
RUN=C:\ibmcom\lanmsgex.exe

```


MAKING IT WORK

Most of the work involved in getting all these protocols running at the same time has already been done by the providers of the protocols. The installation packages do a good job of configuring the system so all protocols load properly and can reside harmoniously in the same workstation. Special tailoring is still necessary, however, for some components.

Each of the following sections describes the steps necessary to install a piece of the network communication protocol support. It is recommended that you install them in this order. If a particular protocol is not needed, skip its section. Although this installation sequence is not required, it makes system configuration much easier.

NetWare

NetWare installation is completed differently depending on which interoperability solution is chosen, Novell's or IBM's. If you choose the IBM

solution, install NetWare according to the instructions provided with the NetWare requester. Then install IBM's LAN requester and protocol support (LAPS). Be sure to select the NetWare protocol, NetBIOS, and 802.2. This adds changes to CONFIG.SYS and PROTOCOL.INI that configure and run ODI2NDI. Figure 4 gives an example of modifying CONFIG.SYS for NetWare support with LAPS. Figure 5 shows the section added to PROTOCOL.INI for the ODI2NDI device driver.

If you choose the Novell solution, the following steps will ensure LAN interoperability:

- Install the NetWare requester for OS/2 2.0 as per its instructions. COEXIST.TXT, a text file on the NetWare requester installation disk, can be printed and used to augment the information in this article.
- Be sure to create a NET.CFG file as indicated in the COEXIST.TXT document. The NET.CFG for a system using T/R is given at the end of this article.



Use This \$100 Coupon To Create Your Own OS/2 Applications Quickly and Easily - No Programming Required!

Now you can use the power of visual programming to build your own custom-tailored OS/2 applications, without writing a single line of code!

ObjectVision for OS/2 lets you create the kind of applications businesses use every day - order entry, time and billing, contact information, inventory management, and more. And you can do it all, simply by using a mouse and your imagination.

And, for a limited time, you can participate in our special **\$100/100%** offer. Save \$100 off the regular purchase price of \$249.95 and be 100% satisfied or receive a full refund, no questions asked. Use the attached coupon, call **1-800-331-0877**, or fax this form to **408-439-9119** before July 31, 1993.

Name: _____

Address: _____

City, State, Zip: _____

Phone: _____ Fax: _____

Payment method: ☐ Visa ☐ MasterCard ☐ AMEX

Card Number: _____

Signature: _____ Exp. Date: _____

OBJECTVISION® 2.0
FOR OS/2®





; Note the "Bindings" lines in each of the
: protocol sections. In each case, the
: commented line is to be used instead of the
: uncommented line when using Novell's
: solution to Interoperability.

[PROT_MAN]

DRIVERNAME = PROTMAN\$

[IBMLXCFG]

IBMTOK_nif = IBMTOK.nif
LANDD_nif = LANDD.NIF
NETBEUI_nif = NETBEUI.NIF
ODI2NDI_nif = ODI2NDI.NIF

[LANDD_nif]

```
DriverName = LANDD$
Bindings = IBMTOK_nif
; Bindings = TOKEN
ETHERAND_TYPE = "I"
SYSTEM_KEY = 0x0
OPEN_OPTIONS = 0x2000
TRACE = 0x0
LINKS = 8
MAX_SAPS = 3
MAX_G_SAPS = 0
USERS = 3
T1_TICK_G1 = 255
T1_TICK_G1 = 15
T2_TICK_G1 = 3
T1_TICK_G2 = 255
T1_TICK_G2 = 25
T2_TICK_G2 = 10
IPACKETS = 250
UIPACKETS = 100
MAXTRANSMITS = 6
MINTRANSMITS = 2
TCBS = 64
GDTS = 30
ELEMENTS = 800
```

[NETBEUI_nif]

```
DriverName = netbeui$
Bindings = IBMTOK_nif
; Bindings = TOKEN
ETHERAND_TYPE = "I"
USEADDRREV = "YES"
OS2TRACEMASK = 0x0
SESSIONS = 100
NCBS = 95
NAMES = 100
SELECTORS = 5
USEMAXDATAGRAM = "NO"
ADAPTRATE = 1000
WINDOWERRORS = 0
MAXDATARC = 4168
```

- When configuring your system to be used with NetWare and any NDIS based protocol, create NET.CFG, containing the lines in the sample NET.CFG, in the root of your boot drive. Modify CONFIG.SYS and make the following changes:

- a. Remove the line invoking the T/R driver installed as part of the NDIS support, most likely named IBMTOK.OS2.
- b. Add a line invoking the ODI NDIS interface driver (ODINSUP.SYS). See Figure 4 for details.
- c. Remove the line calling NetWare's NetBios if one has been installed.

Then modify PROTOCOL.INI to bind to the ODI NDIS interface instead of to the individual MAC driver. See the Bindings = statements in Figure 5 for an example.

OS/2 LAN Requester 3.0

If you are going to install Extended Services for OS/2 (ES) on the same machine as the multiple requester support, you must install ES before installing LAN Services. If you have not already installed the LAPS code for NetWare support, you must do so before installing the OS/2 LAN Requester. Follow the installation instructions provided with the LAN requester to install it and LAPS for OS/2.

Once the requester is installed, reboot the machine to activate the changes made. Even if you do not plan on using OS/2 LAN Services but would like to use any other NDIS-based communications protocol such as Vines or TCP/IP, you must install the LAPS portion of the code so NDIS support is available.

Vines

The Vines requester is installed with Banyan's VCLIENT program. (At the time of this writing VCLIENT was in an incomplete prerelease version.) Make sure that PROTOCOL.INI and CONFIG.SYS are modified properly. See Figure 4 for lines in CONFIG.SYS and Figure 5 for the section in PROTOCOL.INI needed to add Vines support.

TCP/IP

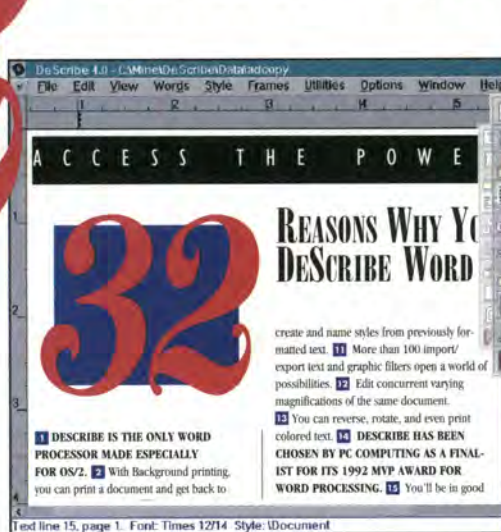
TCP/IP is installed with the installation and configuration tool provided with the base



REASONS WHY YOU SHOULD BE USING DeSCRIBE Word Processor 4.0 SE

1 DESCRIBE IS THE ONLY WORD PROCESSOR MADE ESPECIALLY FOR OS/2. **2** With Background printing, you can print a document and get back to work instantaneously. **3** Character Manager's the easy way to set special character definitions. **4** With the text box tool you can add copy to a picture frame. **5** You can take advantage of DeScribe's interactive definitions dictionary and thesaurus, as well as define and edit the Group dictionary with your unique company or technical language. **6** DeScribe's multitasking and multithreading let you work on multiple documents all at once. **7 BUBBLE HELP ELIMINATES GUESSING AT CONFUSING ICONS AND SYMBOLS.** **8** Master Document Manager lets you control the index, the table of contents, and page numbering across multiple files. **9** DeScribe has always been an innovative leader – and always will be. **10** You can create and name styles from previously formatted text. **11** More than 100 import/export text and graphic filters open a world of possibilities. **12** Edit concurrent varying magnifications of the same document. **13** You can reverse, rotate, and even print colored text. **14 DESCRIBE HAS BEEN CHOSEN BY PC COMPUTING AS A FINALIST FOR ITS 1992 MVP AWARD FOR WORD PROCESSING.**

Undoes document changes you made since you last saved the file.



15 You'll be in good company with a number of Fortune 100 corporations. **16** Mail merge is easier than ever. **17** Automatically print repeating text (like CONFIDENTIAL) on your document pages. **18** Unlimited UNDO restores your work step by step, all the way back to the last time you saved the document. **19** Threaded snapshots automatically back up your work, sparing you from the results of system crashes. **20** Dropped capital letters are a click away. **21** True hierarchical stylesheets exploit an inheritance structure for unsurpassed document formatting. **22 YOU'LL GET A FREE UPGRADE WITH DESCRIBE WORD PROCESSOR 4.0 SE – THE ONLY SUBSCRIPTION EDITION IN THE INDUSTRY.** **23** Both Signature and Two-up printing work together to create booklet formats. **24** Output is easier than ever with drag and drop printing of either individual or simultaneous multiple documents. **25** DeScribe's novice mode simplifies the learning curve.



26 CUSTOMIZE TOOL ICONS JUST THE WAY YOU WANT WHERE YOU WANT. **27** The style palette puts fast dynamic text formatting at your

fingertips. **28** DRAG AND DROP DOCUMENT ICONS MAKE IT FAST AND EASY TO CREATE BOILERPLATES.

29 Only DeScribe Word Processor 4.0 SE gives you unparalleled text control, so you can prepare documents your way – any way you'd like. **30** Creating first drafts is a breeze with convenient draft and outline modes, as well as bulleting and numbering functions. **31** DeScribe Word Processor 4.0 SE is not the first 32-bit word processor – DeScribe 3.0 32 was!

32 DESCRIBE DELIVERS TODAY WHAT OTHERS PROMISE TOMORROW.

DeSCRIBE®

WORD
PROCESSOR

4047 N. Freeway Blvd.
Sacramento, CA 95834
Phone 800 448-1586
FAX 916 923-3447



© Copyright 1992 DeScribe, Inc. All rights reserved.
DeScribe® is the registered trademark of DeScribe, Inc.
OS/2 IGNITED! logo™ is a trademark of IBM Corp. and is
used by DeScribe, Inc. under license. Other brand or product
names are registered trademarks of their respective
companies.



```

TI = 30000
T1 = 500
T2 = 200
MAXIN = 1
MAXOUT = 1
NETBIOS_TIMEOUT = 500
NETBIOS_RETRIES = 8
NAMECACHE = 0
PIGGYBACKACKS = 1
DATAGRAM_PACKETS = 2
PACKETS = 350
LOOP_PACKETS = 1
PIPELINE = 5
MAX_TRANSMITS = 6
MIN_TRANSMITS = 2
DL_CRETRIES = 5

; Remove the following section if
; using the Novell approach to
; Interoperability.
[ODI2NDI_nif]

    DriverName = odi2ndi$
    Bindings = IBMTOK_nif
; It's very important to make sure
; this line is included here. This is
; either the universal address of the
; card or your locally administered
; address, whichever you are using.
    NETADDRESS = "T10005A8FAF12"

    TOKEN-RING = "yes"
    TOKEN-RING_SNAP = "no"
    ETHERNET_802.3 = "no"
    ETHERNET_802.2 = "no"
    ETHERNET_II = "no"
    ETHERNET_SNAP = "no"
    TRACE = 0x0

[VINES_XIF]

    DriverName = NDISBAN$
    Bindings = IBMTOK_nif
; Bindings = TOKEN

[TCPIP_XIF]

    DriverName = TCPIP$
    Bindings = IBMTOK_nif
; Bindings = TOKEN

[IBMTOK_nif]

    DriverName = IBMTOK$
    ADAPTER = "PRIMARY"
    MAX_TRANSMITS = 6
    RECVBUFS = 2
    RECVBUFSIZE = 256
    XMITBUFS = 1

```

Figure 5: Example `PROTOCOL.INI` (continued from page 34)

TCP/IP product called ICAT. To get file redirection with TCP/IP, you must install the NFS portion of the package.

ICAT does not completely install TCP/IP. Two lines must be added to `CONFIG.SYS`, and `PROTOCOL.INI` must be updated to provide binding information for TCP/IP. `CONFIG.SYS` (Figure 4) is modified as follows:

```

DEVICE=C:\TCPIP\BIN\INET.SYS
DEVICE=C:\TCPIP\BIN\IFNDIS.SYS

```

Add the following section to `PROTOCOL.INI` in Figure 5:

```

[TCPIP_XIF]
    DriverName = TCPIP$
    Bindings = TOKEN

```

For performance reasons, it is important that the `INET.SYS` and `IFNDIS.SYS` lines in `CONFIG.SYS` at the end of this document appear before the `NETBIOS.OS2` and after the `NETBIND.EXE` line.

CONCLUSION

Interoperability, an important part of today's network environment, is particularly useful in an environment in which many separate networks exist and in which you want to access all resources simultaneously. Separate logical networks can be connected on one physical network, making all resources simultaneously accessible from a single

```

; This file is required in the root of
; your boot drive if you want NetWare
; interoperability with any of the
; protocols in Figures 4 or 5 and are
; using Novell's interoperability
; solution. If you are using IBM's
; solution, this file is not necessary.

```

```

link support
    buffers 15 4210

```

```

protocol odinsup
    bind token

```

```

link driver token
    frame token-ring
    frame token-ring_snap

```

Figure 6: Example `NET.CFG`

workstation. Most pieces for installing LAN interoperability on an OS/2 2.0 workstation already exist, but they must be properly installed to work together. The example files in Figures 4, 5, and 6 come from a working machine, and provide a tested guide for installation.

REFERENCES

"NetWare Concepts," *NetWare Version 3.11*, Novell Inc. 1983-1991.

"OS/2 Requester Installation," *NetWare Version 3.11*, Novell Inc. 1983-1991.

OS/2 LAN Server Installation Guide (IBM Doc. G326-0162-00)

TCP/IP For OS/2 EE Installation and Interoperability (IBM Doc. GG24-3531-00)

Vines Users Guide For OS/2 2.0, Banyan Systems Inc.

Bob Spratt, IBM Personal Systems Line of Business, 1000 N.W. 51st St., Boca Raton, Fla. 33431. Spratt is a staff programmer responsible for demonstrating OS/2 interoperability at business shows and technical fora. He is also the OS/2 technical contact for outside LAN vendors. Spratt holds a bachelor's degree in computer engineering from Case Western Reserve University in Cleveland, Ohio.



Expressly for the OS/2 Desktop and Workgroup

Relish exploits OS/2 to give you unmatched flexibility in coordinating and managing commitments. Enhancements for the Workplace Shell foster intuitive drag-and-drop manipulation of new and existing calendar entries.

Convenient to use. Print the schedule for any day just by dragging the date to your Workplace Shell printer... Reschedule a meeting by dragging it to a new date or time... It's really that easy.



Relish 32-Bit for personal calendaring brings the intuitive edge to desktop time management. Includes integrated To Do list and Phone Book. \$189.

Relish Net 32-Bit for LAN-based groups blends group and resource scheduling with all the personal features of Relish. Each 5 user increment \$685.

Coming soon... the Relish API. Allows information exchange between Relish and corporate applications. Perfect for scheduling follow-ups. Call for details.

Relish®

32-Bit Time Management



Flexible. Drag-and-drop scheduling and categorizing.
Reliable. Dynamic saving and refreshing across views.
Versatile. Automatic reminders and LAN reconciliation.
Cutting-Edge. Multi-threaded, client-server design.

Ready to spread some Relish? Call for a free taste test. 32-bit for OS/2 2.0. 16-bit versions also available.

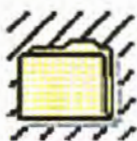
Sundial Systems Corporation (310) 596 • 5121
 909 Electric Avenue, Suite 204, Seal Beach, CA 90740 USA

Introducing
LAN Server 3.0



Welcome

Now all that network computing was supposed to be, can be. Our new OS/2® LAN



Server 3.0 has the functionality, flexibility and high performance to maximize all your network resources. So from individual departments to entire companies, it can make a difference all across the LAN.

Unlike a dedicated NOS, LAN Server 3.0 is built on top

LAN Server 3.0 Advantages:

Reduces Cost.

- Leadership performance.
- Dedicated hardware not required.

Improves LAN administrator productivity.

- Graphical installation tool.
- Remote/unattended installation option.
- Support for remote administration.

Reduces client server application development cost.

- Exploits power of OS/2 2.0.
- Common development platform at client and server.

For additional performance and/or functional information, call 1 800 3-IBM-OS2.

LAN of plenty.

of OS/2. So you get the benefits of pre-emptive multitasking, enhanced OS/2 Crash Protection™ and the Workplace Shell™ GUI. Advanced and Entry versions both support the latest versions of OS/2, DOS and Windows™ on

to the

the
Our Advanced version offers a new High Performance File System (HPFS) that decreases access time to the server's hard disk, improves security and offers fault-tolerant features like disk duplexing and disk mirroring. And LAN Server 3.0 goes beyond simple resource sharing with management enhancements that are systemwide. With sophisticated local and remote system management tools for installation, diagnostics and user security, your LAN will work more efficiently with less downtime. Plus LAN Server 3.0 provides an IBM migration path to OSF's emerging DCE standards.

Token-Ring, Ethernet and IBM PC Network. There's support for Peer Services, optional support for Macintosh computers and TCP/IP, and you can connect more than 1,000 users on a single LAN.

For more information, call 1 800 3-IBM-OS2: OS/2 LAN Server 3.0 does so much more than other network operating systems, it wins by a LANslide.

*In Canada, call 1 800 465-1234. IBM and OS/2 are registered trademarks and OS/2 Crash Protection and Workplace Shell are trademarks of International Business Machines Corporation. Windows is a trademark of Microsoft Corporation. Macintosh is a registered trademark of Apple Computer, Inc. © 1992 IBM Corp.



Tools

Softool: Software Change Management On The PC



by Carlos Caballero

While the integration of OS/2 2.0 into the corporate development environment brings considerable benefit to organizations, it is also creating new challenges in software change management. Software development managers, for example, must ensure that all changes to an application are properly controlled. This requires an internal process to manage the product lifecycle from design through maintenance.

CATEGORIES OF PROBLEMS

Ten years ago, new corporate applications were developed under the control of MIS departments. Today, PC-based development is often done informally, using a variety of hardware and software. Uncontrolled and undocumented modifications to applications or PC configurations can lead to chaos.

The main barrier to effective software change management for desktop systems is a lack of awareness that software changes on the PC need to be managed as strictly as they are managed on the mainframe. Without such management, companies can experience software glitches that can derail system development and maintenance.

Many PC developers tend to focus on their own activities and operating environments without regarding how they affect or are integrated with a larger system. While they have kept up with technological changes, developers do not always see the need to document and control changes to their applications, or to use software change management tools and implementation strategies. Now that the majority of development platforms are interconnected and interactive, individual configuration changes often have large-scale repercussions.



Carlos Caballero

CATEGORIES OF TOOLS

Most software change and configuration solutions on the market today are targeted to narrow audiences operating on a single platform. These tools emerged in the 1970s and 80s, before corporations began moving toward multiple operating systems and interconnectivity.

Several vendors, for example, offer high-end change management and library management products for mainframe use only. Other vendors of PC version management tools provide low to mid-level tools to small business environments. Softool Corporation, in contrast, supports all major software

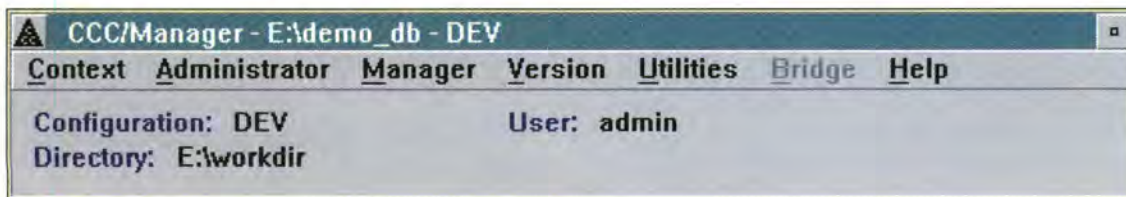


Figure 1: Main window

development platforms including IBM, UNIX, Digital, and the IBM PC.

IMPLEMENTATION REQUIREMENTS

The best place to start planning a change and configuration setup is with a plan of action that spells out corporate objectives and assigns tasks and responsibilities to all personnel involved with PC-based applications, including those in management, development, testing, production, and maintenance. To be

corporate requirements. At the pyramid's peak are cross-life cycle tools that help get the job done.

Ultimately, this implementation approach keeps managers in touch with developers and any changes they make. Managers are also kept informed as to why changes have been made and application status and location. With this setup, it is easy to generate a tracking report that notes all changes.

COMMUNICATION REQUIREMENTS

Communication plays an important role in the effectiveness of change management. The project team must be made aware of the consequences that can result from lack of control and of their role in keeping the system running smoothly.

TOOL REQUIREMENTS

Once the implementation plan is in place, it is time to choose the cross-life cycle tool that is best for the corporation. Because technically-oriented tools are usually limited to developers' use, managers and administrative users running them must rely on developers for information.

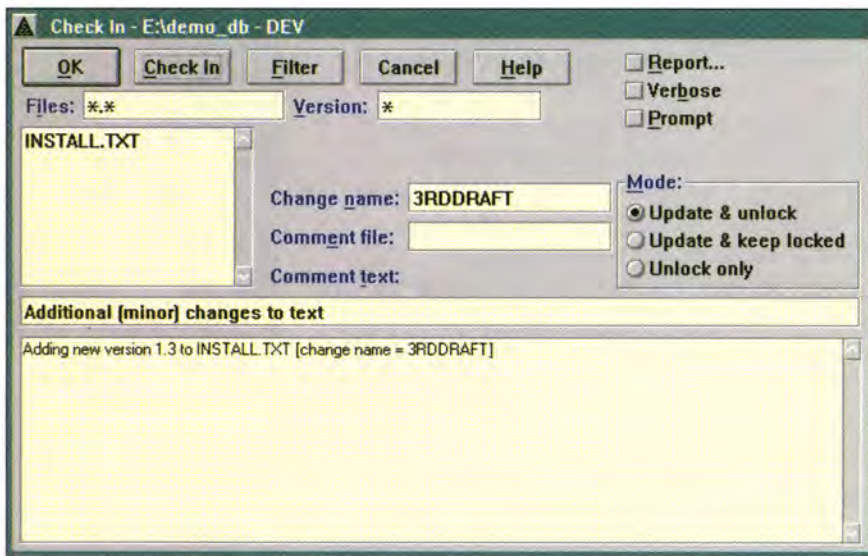


Figure 2: Component management with CCC/Manager

effective, this plan must determine a company's software change management needs, include guidelines and procedures to standardize and automate management, and preferably operate across the entire organization.

The plan can be pictured as a pyramid, with mandatory industry standards at the base. A project team should assess these standards and determine their impact on software development. The goal at this step is to maximize the integration of procedures and tools and optimize cross-tool communication. Next, corporate software requirements, based on the number and types of development platforms in use, must be created to fulfill these standards. The third layer of the pyramid features procedures implemented to meet

CCC/MANAGER

Softool Corp. of Goleta, California has a new product, CCC/Manager™ for OS/2, which provides more functions than older library management tools, while remaining accessible to all participants in the cycle. CCC/Manager supports all life cycle stages from development to testing to production without sacrificing control. It also provides support for the baselining and release processes, change packaging, turnover management, and migration control.

Features

Friendliness—CCC/Manager's user interface, shown in Figure 1, is fully CUA compliant. Everyone involved in the life cycle can use this product.

Component Management—CCC/Manager for the PC lets users store, retrieve, and manipulate versions of any kind of components from sources to executables, spreadsheets, graphics, and so on. Lifecycle components may then be checked out, changed, and checked in, all in a controlled manner. The process of checking an item is illustrated in Figure 2.

Application Management—This layer, unique to Softool CCC™ products, allows users to create "virtual windows" in the repository of versions, each associated with an instance of a given application. Those virtual views or configurations represent the application as viewed from a given life cycle stage, baseline, release, concurrent development area, or customized application version. By automatically updating those views as changes migrate through the life cycle, CCC/Manager releases users from the technical complications and confusion of manipulating error-prone branches or application snapshots.

Package Management—This feature allows users to create, manipulate, and migrate changes in groups called change packages, which contain all elements associated with a problem including documents or spreadsheets. With change packages, changes can be moved as a unit from one staging area to another or promoted to an external directory. Extensive reporting permits management review of the flow of changes. The Package Management layer is shown in Figure 3.

Host Cooperation—CCC/Manager's management of life cycles on multiple platforms involves two phases. In the first, life cycle management must be implemented in all connected platforms, with a version of the product for all platforms in an enterprise. In the second, the control environments are connected to each other with Softool's CCC/Bridge™, an add-on to existing CCC/Manager installations that allows integration of LAN-based and host-based life cycles. Softool's new version of CCC/Manager for OS/2, however, does not require any specific host connection and can run independently on OS/2-compatible workstations and LANs.

SUMMARY

As downsizing increases, mainframes, still the most common host in distributed life cycles, are joined by more and more mini- and microcomputer servers such as UNIX workstations. CCC/Manager for OS/2, configured for multiple environments, covers all these platforms. CCC/Manager is also available for IBM mainframes (MVS & VM), Digital (VMS), UNIX, and Windows.

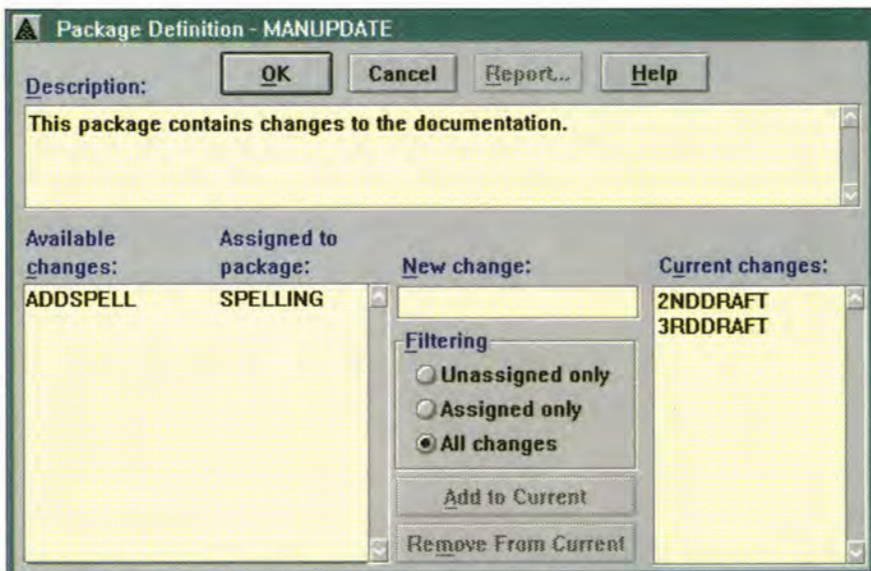


Figure 3: Package Management layer of CCC/Manager

Carlos Caballero, Softool Corp., 340 South Kellogg Ave., Goleta, Calif., 93117, (805) 683-5777, fax (805) 683-4105. Caballero is product manager for Softool. He holds a master's degree in applied mathematics from the University of Central Buenos Aires Province, Argentina, and has more than 18 years of experience managing all phases of the software life cycle.

For more information about Softool's software change and configuration management products, contact Laurie Mix at Softool Corp., 340 South Kellogg Ave., Goleta, Calif., 93117, (805) 683-5777, fax (805) 683-4105.

Stuff Yourself



All the programming you can stand for about \$2 an issue.*

Computer Language serves up serious programming, not just a lot of industry hype and copious amounts of C or Forth code for do-it-yourself toy programs.

You'll get object-oriented analysis, testing and quality assurance methods, techniques for enhancing reusability, effective maintenance strategies, and ways to bridge the gap between design and code. And it all comes from professional software developers who test their skills against programming challenges day in and day out.

But brace yourself for a deluge of information.

Computer Language is not for beginners. It is, however, the only magazine which truly prepares readers to meet the challenges of developing software in the 1990's.

So if you're a serious programmer whose appetite for information has never been satisfied, go ahead... stuff yourself with *Computer Language*!

Clip coupon and send to: *Computer Language*, P.O. Box 53525, Boulder, CO 80322-3525
or call: **1-800-288-1322**



51CL4

☐ **Yes!** I want to receive a full year (12 monthly issues) of *Computer Language* for just \$29.95, a 36%* savings off the single copy price. I understand that **I may cancel at any time and receive a full refund.**

Check one:

☐ Bill me

☐ Payment enclosed

☐ Charge to: ☐ VISA ☐ MASTERCARD ☐ AMERICAN EXPRESS

CARD # _____ EXP. DATE _____

SIGNATURE _____

NAME _____

ADDRESS _____

CITY _____ STATE _____ ZIP _____

* Savings are reflected off the single copy price of \$3.95. The basic subscription rate is \$34.95. Canadian/Mexican orders must be prepaid in U.S. funds with additional postage of \$6 per year. All other foreign orders must be prepaid in U.S. funds with additional postage of \$15 per year for surface mail and \$40 per year for air mail.

Client/Server

Testing Client/Server Applications



by Phil Wallingford

The development of increasingly complex client/server applications is bringing to light the need for more effective software testing and quality assurance. With new GUI applications increasing in complexity, testing costs have exploded; major software suppliers such as Borland International and Lotus report that testers now outnumber developers. Concurrently, the audience for GUI applications is expanding into the millions, multiplying the cost in money and reputation of delivering buggy software.

In the context of these changes, it is ironic that software test technology has advanced little since the beginning of the computing era; most tests are still performed manually. The common myth that software can't be tested needs to be rethought, restated more accurately as "Software can't be tested using a traditional testing approach." It seems that there can never be enough people spending enough time at enough keyboards to assure that a piece of software is ready.

This imprecision can be alleviated through improved testing methods, particularly through automated testing. To be effective, automation must be applied throughout testing, from the test design phase to test development to execution.

This article discusses test automation techniques and tools and their applicability to the special requirements of client/server computing.

SOFTWARE TEST AUTOMATION

A common way to introduce software test automation is to review questions commonly asked of those responsible for testing, such as: "How many problems have you found?" and "How much time have you

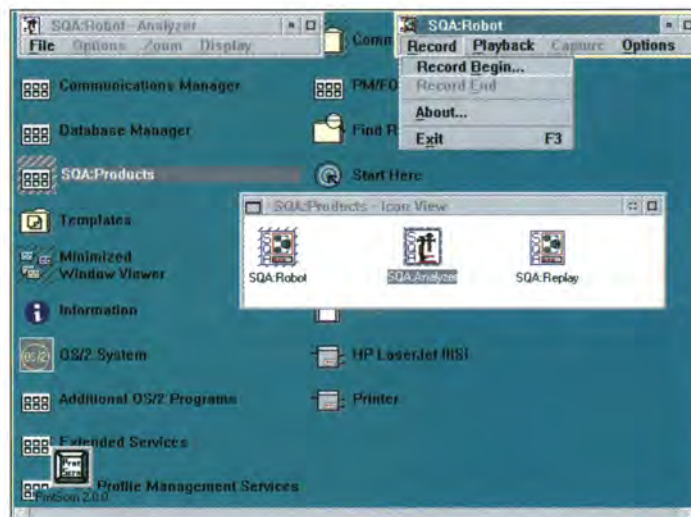
taken to find them?" Unfortunately, although the questioners know that these aren't really the "right" questions, they're the only ones for which there are, typically, straightforward answers.

More helpful questions might include: "How much of the code is tested? If it's not fully tested, when will the testing be done?" and "How reliable is the software? If I field it, what's the estimated failure rate?" These questions can be answered, but not without the application of systematic engineering discipline and automated test tools. Test automation must first be approached by defining the objective criteria used to judge the progress of a test, including what data will be collected and what analysis will be used to determine when testing is complete.

There are two fundamental approaches to data collection in software testing. The first is to collect failure data based primarily on black



Phil Wallingford



SQA:Robot for OS/2

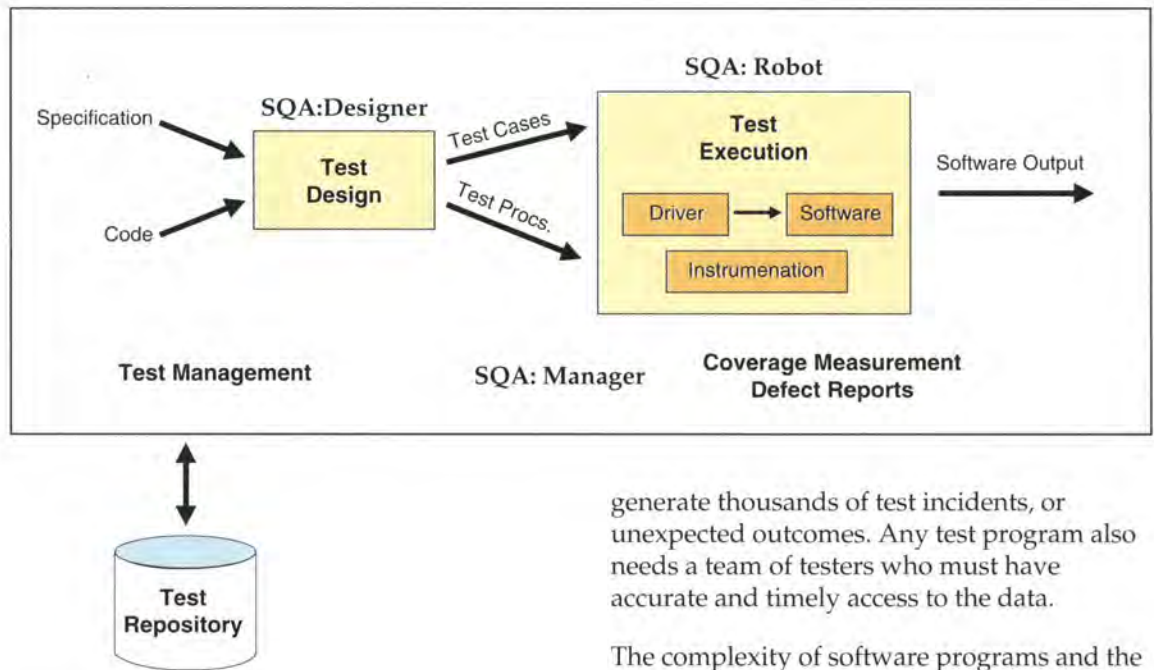


Figure 1: Test automation system

box or functional tests and fit the data to a software reliability growth model to determine the software's readiness for operation. Several different models have been proposed for this kind of data collection, based on the use of different techniques and statistical methods.

The second approach is to establish measures of testing completeness based on some criterion, then collect the associated data to measure progress. In this approach, it is common to use various coverage measures to show how completely the software has been tested. Of particular interest are structural (or white box) coverage measures such as control branch or data flow; functional coverage measures are also common.

Of course, both approaches can be used. Once a tester forms a strategy, he or she must develop an implementation plan, including a set of integrated automation tools. Figure 1 shows the major elements of a system for automating software testing.

Testers must establish a test repository to save test items and measurement data in a structured, retrievable way. The volume of data required to support automated testing is enormous; a modest testing program can require tens of thousands of test cases and

generate thousands of test incidents, or unexpected outcomes. Any test program also needs a team of testers who must have accurate and timely access to the data.

The complexity of software programs and the tools and techniques used to design them makes automation necessary for accurate testing. Test design tools can automate test generation using control or data flow graphing, domain analysis, syntax testing, state graphing, or other structured design processes.

Other tools automate test execution (including capture and playback), collect data quickly and reliably, and save it in the repository. Finally, management tools help maintain the repository, generate reports, and analyze incoming data, answering the questions posed previously.

TESTING REQUIREMENTS

Client/server applications place special requirements on testing methods and tools.

GUI Concerns

GUIs such as the OS/2 Presentation Manager have become an essential part of most client/server applications. These new interfaces, however, place an entirely new set of requirements on testing techniques. New input devices must be recognized; high-resolution spatial and color images must be examined. Most significantly, GUIs bring complexity and freedom to user interaction that makes traditional testing obsolete.

*Automation
must be applied
throughout
testing.*



Reliability

Most client/server applications are developed for critical business functions. An insurance company, for example, may use client/server technology to process its claims payments. Reliability problems in this process can be devastating to the company and its customers.

Unfortunately, the delivery of reliable applications on client/server architectures presents challenges beyond those of traditional host-based applications. In most cases, client/server architectures span multiple processors that are connected through highly layered communications protocols. Some processors may be shared while others are dedicated. This environment, while attractive for its technical merits, is daunting because it is difficult to assure its reliable operation.

Multiuser Systems

Client/server applications serve many users and span large heterogeneous networks. In traditional host-based environments, multiuser system operation can be reliably tested by running user simulators on a central processor. In the client/server environment, the server processor can be tested with a similar approach. The client side, however, demands new testing techniques, especially in regard to the scheduling and monitoring of client software operation. Workstations may be distributed across a wide area without central control, and there is the problem of restarting workstations after fatal errors.

Repository-Based Test Measurement Systems

The high volume of testing required for most client/server applications demands an accumulation and analysis of test data that cannot be handled by manual techniques. Client test systems need electronic access to a central repository to automatically log test results as well as the capability for analysis and report generation to interpret and present them.

Concurrency

Access to shared resources must be efficient and reliable. Most client/server applications share a database, with workstations sharing

responsibility for maintaining the integrity of system resources. Workstations have been providing multiuser concurrent processing for a relatively short time. Testing techniques must verify their ability to perform this processing. In addition, the ability of a client workstation to manipulate database records locally creates new opportunities for concurrency errors.

Performance

Users of client/server applications expect performance at least equivalent to that of familiar host-based applications. The flexibility of client/server computing's multiprocessor setup, its great strength, will require new ways of monitoring and tuning performance. Within the workstation, as new OS/2 applications are delivered and enhanced their code segmentation structure will need tuning. This requirement, familiar to large system developers, must be adapted for workstations.

```
MouseDown (230, 245)
MouseMove (230, 248)
MouseMove (234, 256)
. . .
MouseUpn (321, 420)
etc.
```

Figure 2: Low-level script

```
FocusOn ("Editor")
MenuPick ("File->Open")
TypeKeys ("File1")
etc.
```

Figure 3: High-level script

TOOLS AND TECHNIQUES FOR TESTING

Software Quality Automation supplies test automation tools for GUI and client/server development. From experience assisting customers in delivering major client/server applications, the development staff has distilled a set of basic requirements that address the special needs of client/server testing.



Test Execution

Test-execution tools are the most ubiquitous of all test automation tools. They automate the application of test cases to the software under test and provide data about software performance. Capture/playback execution tools, the most common, capture a user's keyboard and mouse operations into a script, which is then played back on a later software release to verify consistent operation. This test

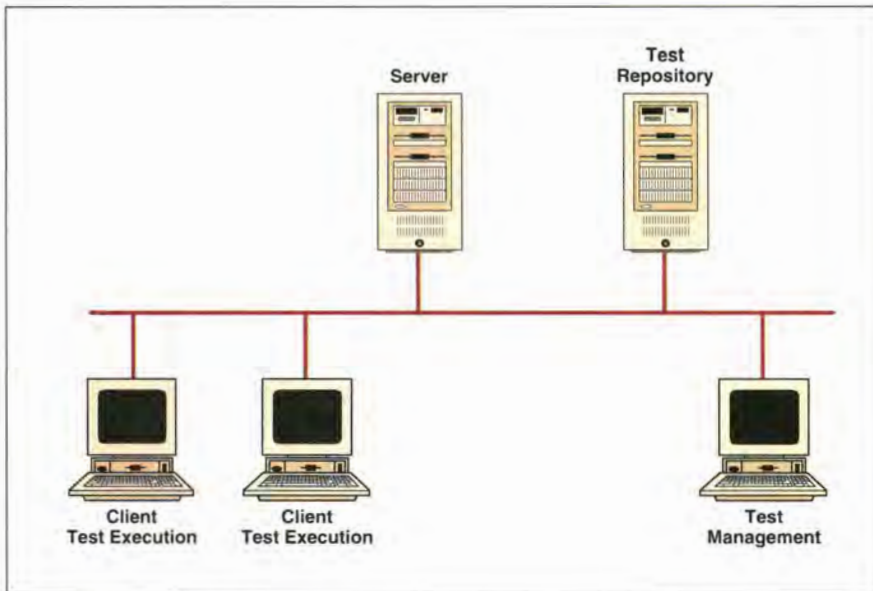


Figure 4: Multiuser testing

approach is called "regression testing" because software is checked to ensure that it has not regressed to an earlier state. Capture/playback tests are usually black-box tests performed during a system test.

Traditionally, test scripts are saved in some proprietary format and are often uneditable. However, test scripts must be adapted to changes in the software under test. In tools that allow testers to edit scripts, the scripting language is also proprietary. The language is also usually low level, dealing, for example, with mouse movements that are difficult to interpret, let alone modify, as shown in Figure 2. These factors combine to make script editing tedious at best.

Newer capture/playback tools, such as SQA's SQA:Robot™, have advanced scripting by first recording the script in a standard scripting

language and translating the low-level captured information into high-level operations, as shown in Figure 3. A scripting language should be integrated with a standard programming environment such as BASIC, C, or REXX. Building the scripting capability on a standard environment brings the power of a full programming tool to test scripting, reduces training requirements, and enables testers to integrate third-party software packages.

The scripting language also becomes the vehicle for delivering automated tests generated not by a user's actions but by automated test generation techniques. Built-in test cases allow testers to compare screen images, examine output data, verify file or window existence, monitor interprocess communication, and check file contents. They can also integrate user-defined test cases into the scripts. Finally, wait states allow the software under test to stay synchronized with system resources such as network traffic.

Similarly, testers can track system performance and monitor system resources such as memory, stack usage, and disk space with test execution tools. They can measure performance of software under varying system configurations. Coupled with test design tools that analyze the structure of the software, test execution tools can also capture test coverage information such as statement, branch, path, and data flow.

Test-execution can be expanded to include network testing. In a client/server environment, multiuser testing runs test scripts simultaneously on a number of client workstations. (Script execution can be scheduled with any network scheduling software.) Test results, including performance characteristics, can be accumulated and monitored on a server, as in Figure 4.

Error Recovery

Error recovery for a client workstation is essential for productive client/server testing. Error recovery should be available on at least three levels. First, the scripting language should allow the developer to detect test case failures and take appropriate action (for example, skip to the next test case). More serious or unexpected faults, such as

Get The Big Picture. One Piece At A Time.

SOFTWARE

Graphical User Interface Design

Heterogeneous Networks

Interoperability

Coding To Standards (X/Open, POSIX, ANSI, etc.)

P.J. Plauger

Desktop UNIX

X Windows: Motif and OpenLook

Security

Object-Oriented Databases

Distributed Computing

Programming in C++

Real-World Open Systems

CASE

Rightsizing

Jon Bentley

Cross-Platform Development

Programming Environments

DEVELOPMENT

1993 CONFERENCE & EXHIBITION

February 21-26, 1993 - Santa Clara, California

Andrew Binstock

For those of you sifting through dozens of conference invitations, Software Development '93 has something you might like: everything.

If you want to learn how to develop desktop systems cleanly, quickly and cost effectively, or if you manage the efforts of others, fax us this coupon or give us a call. We'll tell you how to get the big picture a piece at a time at SD '93.

YES! I Want More Info on:

☐ Attending ☐ Exhibiting



Name _____

Title _____

Company _____

Address _____

Phone _____

Fax _____

Software Development '93 600 Harrison St. San Francisco, CA 94107
Phone 415-905-2741 Fax 415-905-2222

UR



protection violations or stack overflows, must be trapped so the system can reset to a known state before proceeding. Finally, the test environment should be able to recover from fatal system crashes with the aid of a hardware timer and a bookmark facility (which identifies the point of failure) in the test scripts.

Test execution tools currently address testing of the user interface. The loosely coupled architecture of client/server applications suggests other approaches, elaborated in Figure 4. Application building tools can define objects with well-developed APIs, while the client/server interface can be specified at different levels, such as remote procedure calls or SQL calls. Each of these interfaces is an attractive "hook," or connection to monitor the software for capture/playback technology.

"Adequate" must be defined, raising the question of how to determine when enough testing has been done.

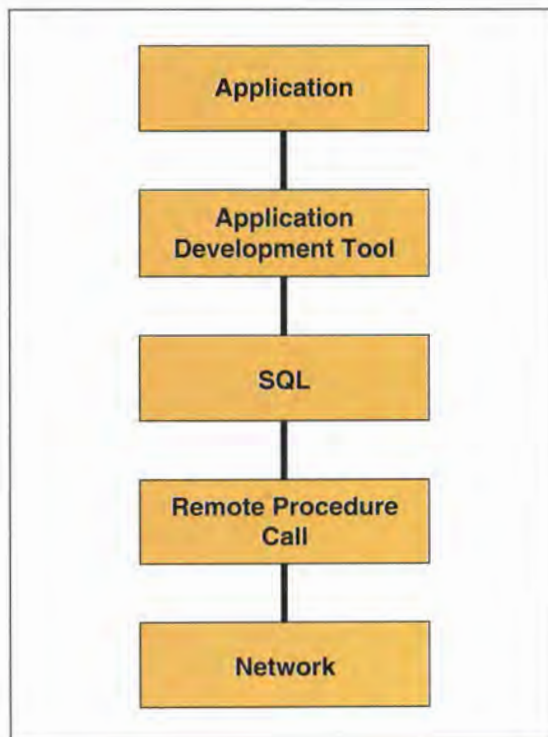


Figure 5: Client/server APIs

Test Design

Currently, test design tools are the least used automation tools, a sign of the immaturity of testing methods. (In fact, test design itself appears to be absent in many instances, as evinced in the saying "Build the test first; design it later.")

Test design tools will eventually become the most important elements of the tool set, removing the last large labor component in test suite design. They are also the only instruments that can generate the large numbers of tests, such as data sets, that will soon be required for adequate testing.

But "adequate" must be defined, raising the question of how to determine when enough testing has been done. Commonly, the definition is based on available testing resources (such as testing until the promised delivery date), with little regard to fundamental measures of software quality. More scientific measures are based on criteria such as branch and path coverage, data flow coverage, and reliability. Good test design defines the test cases that provide these measures.

Test design can be addressed to either functional or structural (typically unit) tests. Design automation tools for functional tests are hampered by a lack of formal software requirements or functional specification languages. As a result, test coverage must be defined by interpreting natural language descriptions. Testers can use automation tools to catalogue and organize the test coverage points and associated test cases to generate coverage completion reports.

Design automation can also be applied to recorded test scripts (generated with a capture/playback tool, for example), especially when the scripts exist as a high-level language. With a script recorded to populate cells in a spreadsheet and verify the results of calculation, an automation tool can generate hundreds of variations of input-output data sets (test cases) that can be read by the script during playback. With recorded scripts, it is best to describe inputs and outputs in a formal syntax (such as Backus-Naur form) and apply a general purpose test data generator.

Structural testing lends itself to automated test design, as the software code exists as a formal description of how the software operates. (How it operates is not necessarily the same as how it is meant to operate, but that's another subject.) With this formal description as a foundation, structural test design can become highly automated.

Look who came back from Vegas a winner.



These leading computer publications have just awarded OS/2® 2.0 top honors at Fall COMDEX. Which confirms what we've been saying all along. Quite simply, OS/2 is the most complete, most reliable, most well thought out PC operating system you can buy.

When you're in Vegas with a system like that, you can't help but win. To find out more about OS/2, call 1 800 3-IBM-OS2.

IBM®



One test coverage approach, relatively easy to implement, produces important information for performance tuning of client applications. Function coverage (as in C functions) mapping provides a reasonable, although sparse, means of test coverage. Testers can devise a strategy for optimizing (segment swapping) performance from the generated function call tree profile. This tuning will become increasingly important in client/server computing as applications take advantage of virtual memory systems on the client workstation.

Test Documentation

- Test plans
- Test design specifications
- Test procedure specifications
- Test case specifications

Test procedures (scripts)

Test cases

- Benchmark results

Test logs

Failure data

- Incidents
- Problems

Reference information

- IEEE, DOD, and ISO standard documents
- Corporate standards

Figure 6: Test repository contents

Test Repository

The Test Repository, at the core of the test system architecture, is a central facility used to store, organize, and validate all information necessary to support a robust departmental (and ultimately corporation-wide) testing and quality assurance program. The Repository also provides access to existing source-code libraries, enabling automated static analysis, code metrics, structural test case generation, and so on. Examples of information maintained by the Repository are shown in Figure 6.

The central technology in the Test Repository is a robust database management system. All

test data is cross-referenced through this facility to maintain configuration control in an evolving product release cycle. For example, a tester can make queries to determine which test cases will have to be reviewed if a particular software module changes. The Repository also supports group testing efforts with security and record-level locking features.

SQA delivers all of its products integrated with a Test Repository, which all test automation tools use as their information back plane. The Repository's data model is fully disclosed; it is accessed with published access methods.

Test Management

Managing a software test and quality assurance program requires a system to measure progress and analyze results. The system should free testers from tedious clerical tasks. The management system should be integrated with design and execution tools so all can access common data. Test management tools maintain the Test Repository and analyze and report on stored data, giving quality assurance, test, development, product management, and customer service staffs a unified view of the status of a testing program. Toward this goal, SQA has delivered the industry's first PC-based test management system, SQA:Manager™, which is fully integrated with SQA's Test Repository and other automation tools.

SUMMARY

As development organizations build client/server systems, testing practices need to be reviewed and adapted to automated techniques. Test automation is becoming essential to deliver reliable client/server applications.

Phil Wallingford, *Software Quality Automation*, 1 Parker St., Lawrence, Mass., 01843, (508) 689-0182. Wallingford founded *Software Quality Automation* in 1990. He has over twenty years experience in software development and has led development teams working on CAD/CAM product development and multimedia applications. Wallingford can be reached at the address above or on CompuServe.

Not enough COM lines on your computer?

Then expand!

Use an IBM ARTIC co-processor card and Quadron software. You'll gain turnkey access to multiple async communication lines, and the processing won't jam your computer because it's done on the ARTIC.

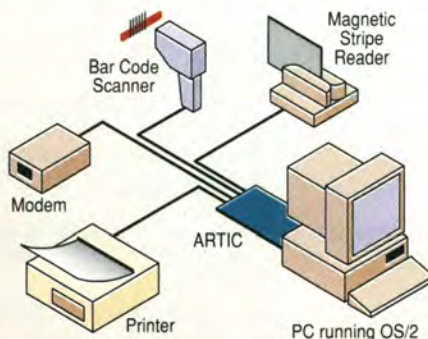
Can I run LAN Manager RAS?

You'll be able to support multiple lines for Microsoft's LAN Manager RAS (Remote Access Service), as well as for cc:Mail's Router, or for dial-in bulletin board systems like Magnum or Multinet.

Do I have to change my software?

If your application now runs on regular computer COM ports, it will work the same

way on ARTIC cards with Quadron communication software.



With Quadron software and ARTIC cards in the computer expansion slots, up to 32 COM lines can serve user needs.

Why an IBM ARTIC card?

Manufactured by IBM's Industrial Division, ARTIC cards are industrial-strength workhorses, supported by IBM worldwide, and built for year-after-year reliability.

So call us today.

We've got ARTIC cards and software on the shelf right now. Phone us for details.



Quadron

Quadron Service Corporation
209 East Victoria Street
Santa Barbara, CA 93101
805-966-6424

© 1993 Quadron Service Corporation. All product names are trademarks or registered trademarks of their respective holders.

How much time do you waste searching for bugs?

Toolkit API calls usually work. So that you don't have to check every return code, Error Manager does. EM tells you which API failed, where it failed and why. EM can log all errors and your messages to a file, pipe or to our viewer. And, it optionally notifies your window procedure enabling centralized error processing.

Error Manager



32-bit
Version 2.0
for OS/2 2.X
\$225

- Invaluable for Debug, QA Test and Production cycles.
- Finds intermittent errors without the Debug Kernel.
- Works with optimized code in realtime situations.

----- Session started on 11/2/92 at 4:11:14 -----
Programmer Message - This is my message 1
WinGetPS() generated error 1001 - Invalid window handle
in file TEST.C at line 19.
DosRead() generated error 6 - Invalid handle
in file TEST.C at line 21.
Programmer Message - This is my message 2

Include 1 header file.
Link 1 DLL.
Set 1 environment variable.

Includes a PM Viewer,
Pointer Validity API and a
Free copy of Command Line

Soft & GUI



2224 East 21st Street
Brooklyn, New York 11229
(718) 769-8017

Tennis Pros, Golf Pros And OS/2 PM Programmers Have One Thing In Common...

...they need proper training and tools to be successful !

The right mix makes all the difference! PM programming is easier than you think IF you use proper object-oriented techniques and high level PM window classes.

Ask for SE International's

- ✓ **Inhouse PM programming training.** Within one week you learn how to succeed in large C and PM projects and how to choose the right development tools.
- ✓ **PM Extensions.** Powerful new 16-bit and 32-bit PM window classes (**FormattedEntryFields**, **DateEntryFields**, **PrimaryWindowClass**, **Animated Bitmaps**, **MultiColumnListBoxes**...) allow you to focus on the application logic and not on programming of frequently required window behaviors. All our **PM Extensions** are **Gpfl™** compatible.

SE International, Inc.

621 NW 53rd Street, Boca Raton, FL 33487
Tel: (407) 241-3428 • Fax: (407) 997-8945

SES Software Engineering Services GmbH Berlin

Koenigsallee 49, D-1000 Berlin 33, Germany
Tel: +49 30 826 40 63 • Fax: +49 30 825 30 14

OS/2 PRODUCTS & SERVICES

TCP/2™ tcp/ip for OS/2

"TCP/2™ is, by far, the best implementation of the tcp/ip application suite for DOS and OS/2 available from any source"

Microsoft ITIS

Essex Systems, Inc.

One Central Street Phone (508) 750-6200
Middleton, MA 01949 Fax (508) 750-4699

BUILD YOUR OWN OS/2 LIBRARY

Reprints are now available for **OS/2 DEVELOPER**.

Use your customized reprints for: **Customer support, sales tools, marketing support, educational tool.**

Call today for a custom quote: Andrea Varni, **OS/2 DEVELOPER**
415-905-2552 or fax 415-905-2236.

SUBSCRIBE TODAY!

Only \$39.95 for a full year subscription to the premier source of tools and techniques for the OS/2 software developer: **OS/2 DEVELOPER**

Call today and don't miss a single issue:

1-800-WANT-OS2

Attention Mathematica users:
Everything you need to know to get the most out of your math programming is in

THE MATHEMATICA JOURNAL

Call or fax today for information about subscriptions and special issues:

Phone orders: 415-905-2332

Fax orders: 415-905-2233

NEURAL NETWORK **SPECIAL REPORT**

Your comprehensive guide to neural network tools and the companies that make them - all new for 1992!

All for only \$7.95.

To order, call 415-905-2376, or fax to 415-905-2233 now

BUILD YOUR OWN OS/2 LIBRARY

Reprints are now available for **OS/2 DEVELOPER**.

Use your customized reprints for: **Customer support, sales tools, marketing support, educational tool.**

Call today for a custom quote: Andrea Varni, **OS/2 DEVELOPER**
415-905-2552 or fax 415-905-2236.

MISSING SOMETHING?

Complete back issue library available today!
Every back issue of OS/2 DEVELOPER can be yours.

Call today and complete your library:
1-800-WANT-OS2

Client/Server

Testing in the GUI and Client/Server World



by Marvin Tener

It is no news to OS/2 developers that testing in a GUI environment is a difficult task. As developers, we are now at the point where our ability to build systems far outpaces our capacity to reliably test them. At the same time, there is increasing pressure on commercial product developers and corporate MIS departments to produce higher quality software. If there is one thing being hammered home again and again, it is that users have had it with serving as unwitting beta sites.

GUI client/server applications are becoming more and more popular with users and developers. Such applications are, after all, powerful, flexible, and easy to use. Ironically, though, the very features that give GUI client/server applications their appeal are what make them so difficult to test.

THE GUI CHALLENGE

With GUIs, what the user perceives as freedom—moving a mouse, clicking on icons, opening multiple windows—the software tester experiences as random behavior, difficult to model and predict. The tester, moving a mouse across the desktop and clicking, is obscured from the workings of the operating system and from what the application sees in relation to what he or she is doing. The tester has little control over certain actions, such as windows repositioning themselves and coming up in new areas on the screen.

The entire model presented by GUIs is far different from the hierarchical model available in the character-based world. To reach a screen at which data entry will occur in the DOS and mainframe environment, a user has to wade

through several levels of menus. In this environment, an application takes a piece of information from the screen (for example, the highlighted first letter of a menu pick), performs a task, completes it, and grabs the next piece of information when the application is ready for it.

Applications under GUIs are event driven, with applications responding to events that can come from anywhere: a keyboard entry, mouse click, dynamic data exchange, object linking and embedding, information from another application, and so on. The difference between character-based and event-driven processes has been termed "pull-push," with the character-based model pulling in information and the event-driven model having information pushed at it.

In determining what this means for an application tester, it may be clearer to define the difference using baseball as a metaphor. In the character-based world, the application is the pitcher, controlling the tempo of a game by throwing the ball when it is ready to do so. In the event-driven world of GUI, the application is the shortstop, scrambling after balls hit in its direction. Admittedly, this analogy is imprecise: a pitcher can toss a wild pitch or get hit by a broken bat, and a good shortstop will have some sense of the hitter, game strategy, and so on. On balance, however, it is the pitcher who more often controls the game and whose activity is easier to predict and prepare for, and the shortstop who must remain alert for possible action.

Under GUI, there is the added complexity surrounding available system resources. Testers in the DOS and mainframe world are mainly concerned with disk space, file



Marvin Tener

What the user perceives as freedom—moving a mouse, clicking on icons—the software tester experiences as random behavior.



handles, and memory. Beyond that, there are limited resources to worry about. In the GUI world, there are additional resources—such as pens, window handles, and device contexts—that may be in use by one or more applications. A GUI tester must consider many more complex resources.

THE CLIENT/SERVER CHALLENGE

Testers now need to spend more time testing the interaction of application focus points with the user interface, operating system, and network.

With networked applications, the testing landscape becomes more difficult as applications are moved from the controlled mainframe environment into the more open world of the client/server domain. It is no longer sufficient to test the unit functionality of software, but is now necessary to determine how it will act in a complex environment that may contain disparate applications running under multiple versions of operating systems across multiple platforms.

In addition to the complexity of testing itself, there is the additional issue of fixing a problem once a system has been installed. In the "old days," it was relatively easy for an MIS director to perform a test run when development finished with the executable, which, after all, would reside in just one place. If a problem occurred, it was relatively simple to roll back to a stable version of an application. In the client/server world, in which applications may be distributed over thousands of machines at many different sites, the actual delivery of the system to those machines is a physically daunting task. If there is a problem, it is harder to roll back to a stable version of an application.

THE OPERATING SYSTEM CHALLENGE

Intertwined with the GUIs and networks on which they run are the powerful operating systems that hold everything in place, which can also present problems for testers. In the DOS world, operating systems were thin in their support, offering little beyond the ability to read and write files, allocate memory, and put a character to a screen. Those charged with testing an application were involved primarily in testing the code that was being generated, with little attention paid to the underlying operating system or to the user interface.

OS/2 does a lot more than DOS, and is far more robust. OS/2 2.0 also offers a range of features for developers: standard controls, multithreading, resource management, elaborate user interface support, and network and mainframe communication facilities. In addition to testing basic application code, testers now need to spend more time testing the interaction of application focus points with the user interface, operating system, and network.

TESTING IN THE DEVELOPMENT CYCLE

As applications have become more complex, the role of those responsible for software testing has grown to that of "quality assurance professional." In mainframe development, the rule of thumb was one quality assurance person for every five to 10 software engineers. We are now seeing a marked shift in these ratios. At the Software Testing Analysis & Review conference in Las Vegas, Nevada, in May 1992, David Moore of Microsoft's Worldwide Product Division noted that in his division there are two quality assurance professionals for every three developers; at some points during the 1-2-3 Windows™ release at Lotus, there were two quality assurance professionals for every developer. These dramatic shifts have occurred among both independent software vendors and large corporations.

As testing grows in importance, tools that assist quality assurance are emerging. New tools that address software testing and maintenance represent a change from those released during the 1980s that focused primarily on the application analysis, design, and coding aspects of the application development cycle, such as CASE, fourth generation languages, C++, and object-oriented programming.

TEST PROCESSES

Softbridge currently produces a software testing product, the Softbridge Automated Test Facility™ (ATF) for testing stand-alone and client/server OS/2 and Windows applications. Among the first to develop large scale GUI client/server systems in the mid-1980s, Softbridge worked with IDS/American

An Automated Regression Testing Tool for OS/2 PM

PMTESTER

- o Test OS/2 PM and OS/2 Windowed applications
- o Test Windows applications on OS/2 2.0 Workplace Shell
- o Screen Capture, Comparison and Masking
- o Real-time "Smart" playback of Test Cases
- o Logged Test Case outcomes, start/end time and duration
- o Batch execution of Test Groups and Test Cases
- o Adjustable execution speed

And many many more features for only \$595.00!!!
Runtime version also available for \$200.00.

For a limited time only
SAVE \$100 SAVE \$100
on PMTESTER Version 3.10 package for \$495

Princeton Windowing Systems, Inc.
P.O. Box 7635, Princeton, NJ 08543-7635
(609) 921-6695

CPU MONITOR BOOSTS OS/2 TO THE MAX

CPU Monitor™ gives a real-time boost to your OS/2 1.2, 1.3, and 2.0 environment. With CPU Monitor, you can display statistical data for all processes, threads and estimated CPU utilization; detect and stop

invisible, runaway, and background programs; suspend and resume threads, and dynamically change thread priorities for any of your Presentation Manager applications. You can tailor the displays exactly to your liking, since CPU Monitor's displays are fully configurable.

For a limited time,
CPU Monitor is only
\$ 99.95 Start
getting maximum performance out of OS/2.
Order CPU Monitor today.

**BonAmi
Software Corp.**

60 Thoreau Street,
Suite 219
Concord, MA 01742
(508) 371-1997

First there was the Pony Express...



**NOW... Announcing
Icon EXPRESS for OS/2 2.0**



*Change icons on the
desktop in one simple step!*

Icon EXPRESS Utility for use with OS/2 2.0:

Fast... Changes icons on the desktop with a click of the mouse. Instead of 12 steps to change an icon, **Icon EXPRESS** reduces the number of steps to 1! Simply choose the icon to be changed, drag it to the new icon and click!

Simple... Any icon on the desktop can be changed. **Icon EXPRESS** copies any icon to the clipboard for use in other applications.

Organize... Manages other icons in your system. To organize your icons, simply create a new page in the **Icon EXPRESS** notebook and bring icons from other applications into these pages. You now have easy access to all of your icons.



Includes Super 1000 Icon Library:

Computer Devices	Documents	Folders
Business/Office	Sports Figures	Nature
Letters of the Alphabet	Geometric Designs	Tools

**INTRODUCTORY
PRICE ONLY: \$49.99**



**To Order: 1-800-ACT-7185
American Computer Technologies, Inc.**



Express to build a support system for its financial planners. The initial system was a three-tiered architecture made up of DOS-based front-ends, DOS application servers, and mainframes. Over time, the system grew to a point at which OS/2 replaced DOS as the front-end and server operating system. During implementation of this system and others for Nedlloyd Lines and MCI, Softbridge encountered problems that led to the development of ATF. In doing so, the company came up with working requirements for a testing tool: consistency, repeatability, unattended operation, and networking ability.

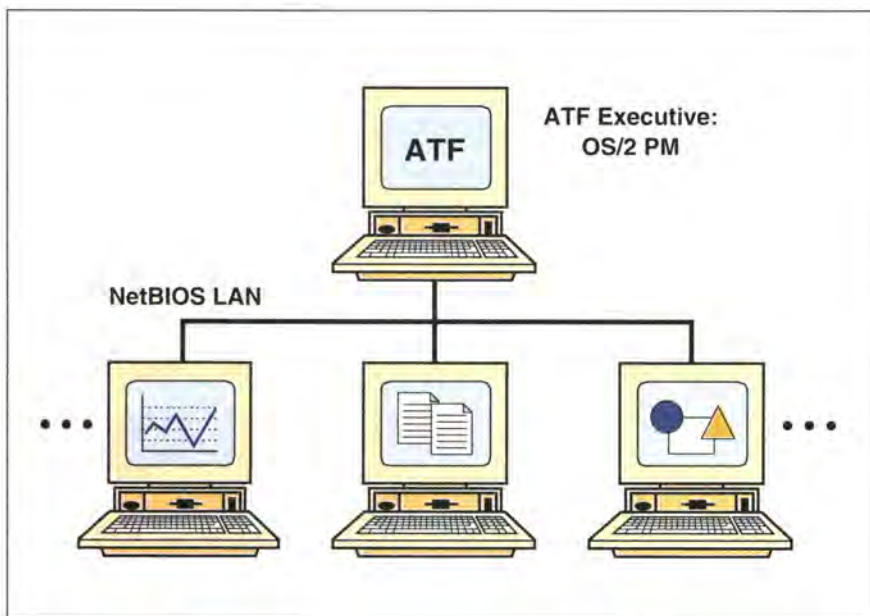


Figure 1: ATF architecture

Consistency

Testing consistently means being able to codify or script a test case, either through a scripting language or through capture/playback recordings of user interactions with an application. The best-case scenario is a testing product that provides the capability of blending tape recordings and programmatically created scripts. Capture/playback allows testers to quickly create a set of test cases. Programmatic development of scripts lets testers develop generic test cases that can evolve over time and build test cases before software is finished.

If tests are done manually and a fault or an error occurs, it is difficult for testers to

remember exactly what they were doing at the time of the error, unless they were following an explicit script. (This pretty much leaves out using the mouse or behaving like a normal GUI user.) With scripted test cases, testers can present an exact record of their actions.

Repeatability

Studies have shown that during the release of an application, software moves between development and quality assurance an average of six to 15 times. As the release date nears, considerable pressure is placed on testers to restrict their activity to areas where change has occurred. Sound testing practice calls for running a full set of regression tests on an application each time a change is made, identifying effects that cannot be uncovered if only the changed code is fully tested. With automation, once you incur the startup costs of codifying the test suite—admittedly, a more time-consuming process than simply running through the tests manually—testers are guaranteed the ability to run their entire test plan no matter where they are in the development cycle.

Unattended Operation

As a measure of testing thoroughness, code coverage (that is, the percentage of code in an application that has been tested) is not as important in the GUI world as in the mainframe world. Considerable state information that cannot be captured through code coverage is incorporated into GUI-based operating systems. Many testers are shifting to mean time to failure to judge the completeness of testing. Unattended testing facilitates longer-term tests to determine mean time to failure, uncovering obscure problems that might not appear in the field for many months or might be difficult to replicate.

Networking Ability

Only a networked test tool can easily replicate real-world client/server conditions. Networked testing can coordinate client/server activities such as testing for load conditions or lock-out when 20 operators are using a system simultaneously. With networked testing, you can also modify an environment by changing screen resolutions, memory configurations, or drivers.

SOFTBRIDGE AUTOMATED TEST FACILITY

The Softbridge ATF was designed to satisfy the previous four criteria. It consists of two major software components, the Executive and the Controller, that mirror the client/server environment. With its Executive, ATF provides a command center from which testers can view and control the operation of complex distributed applications on up to 50 target machines driven by ATF Controllers, as shown in Figure 1. Under the direction of the Executive, an OS/2 Presentation Manager application running under either OS/2 1.3 or 2.0, ATF can run simultaneous tests on OS/2 or Microsoft Windows applications, with the Controllers acting in concert to run the applications being tested.

ATF can also drive stand-alone applications on multiple hardware platforms simultaneously, for increased testing throughput. It is this one-to-many configuration that allows users to implement comprehensive test cases that simulate field conditions for client/server applications. The number of complex tests that can be implemented with ATF far exceeds those that can be orchestrated with human testers. This is because ATF's architecture provides a flexible framework for complex configurations that allow users to increase the volume of tests they can perform, to orchestrate tests of multiuser applications, and to isolate tests to optimize their ability to run unattended.

ATF Executive

The multitasking ATF Executive is an integrated test control center from which tests on target machines (Controllers) are governed. Test scripts are written at the Executive and run against the applications being tested on the remote machines. The Controllers then report test results to the Executive, where they are reviewed and analyzed with the Results Viewer.

Distributed Test Language (DTL)—The core of the ATF system and the key to ATF's ability to run tests consistently and repeatedly, DTL is the language ATF uses to codify test cases into executable test scripts that restore data to a given state, run tests from a central point of control, and save test results in a central log file.

Tape Capture/Playback—ATF records and, on playback, recreates a user's keyboard and mouse interaction with the applications being tested. The context-sensitive recorder moves, sizes, and repositions icons and windows ahead of playing tapes. ATF can play tapes back at variable speeds, particularly useful when a test plan calls for running large quantities of data against a system as quickly as possible. In areas such as stress testing, this can save hours. ATF's tapes can also be parameterized and merged with data to



```
# Connect to workstation(s) under test
connect gsos2

#Start application to test —the OS/2 Systems Editor—
appstart e

#Select option from menu bar
menuselect(app,"File|New")

#Enter text
play keys "Softbridge, Inc."
play keys "<enter>"
play keys "ATF Means Quality Software"

#Select option from menu bar
menuselect (app,"File|Save")

#Enter text to name the file being saved
play keys "SBATF"
play keys "<enter>"

#Exit the application
appexit e
```

Figure 2: SCRIPT

```
#OS2: Sets the Environment
SET SYSTEM TO OS2
SET COUNTRY TO US
SET CAPSLOCK OFF
SET SCROLLLOCK OFF
SET NUMLOCK OFF
SET MOUSEBUTTONS TO 2

#Sets up the Desktop
WINDOW E.EXE
MODULE "E", CLASS "EHXMAIN"
TITLE "e.EXE - Untitled"
END WINDOW
```

Figure 3: TAPE (continued on page 58)



```

WINDOW DESKTOP_MANAGER
  MODULE "PMEXEC", CLASS "StarterWindow"
  TITLE "Desktop Manager"
END WINDOW
WINDOW GROUP
  MODULE "PMEXEC", CLASS "StarterWindow"
  TITLE "Group - Main"
END WINDOW

WINDOW ATF
  MODULE "EXEC", CLASS "ATF_MASTER"
  TITLE "ATF"
END WINDOW

#Resizes the windows and icons
SELECT E.EXE
SIZE NORMAL (76,152) TO (573,472)

SELECT DESKTOP_MANAGER
SIZE ICON (539,36) TO (571,68)

SELECT GROUP
SIZE ICON (439,36) TO (471,68)

SELECT ATF
SIZE ICON (239,36) TO (271,68)

SELECT E.EXE
ACTIVATE
DELAY 0, MOVE (25,35)      #Mouse movement
DELAY 0, LEFTDOWN (25,35)  #Mouse button down
DELAY 0, MOVE (25,35)
DELAY 17, MOVE (25,37) (26,39) (26,41) (26,41) (26,42)
DELAY 9, LEFTUP (25,36)    #Mouse button up
DELAY 5, "Softbridge Inc." #Entering Text
DELAY 15, "<Enter>"
DELAY 5, "ATF Means Quality Software"
ACTIVATE
DELAY 53, MOVE (27,31)
DELAY 0, LEFTDOWN (27,31)
DELAY 0, MOVE (27,31)
DELAY 0, LEFTUP (27,31)
ACTIVATE
DELAY 54, MOVE (32,90)
DELAY 0, LEFTDOWN (32,90)
DELAY 0, MOVE (32,90)
DELAY 0, LEFTUP (32,90)

SELECT SAVE_AS      #Saving the file
SIZE NORMAL (82,128) TO (567,427)
DELAY 122, "S"
DELAY 12, "B"
DELAY 5, "A"
DELAY 4, "T"
DELAY 5, "F"
DELAY 23, "<Enter>"

```

Figure 3: TAPE (continued from page 57)

generate tests for field oriented applications. All tapes are fully editable.

Result Viewer—The result viewer is a Presentation Manager application used to view result logs and window text files.

Bitmap Viewer—The bitmap viewer is a Presentation Manager application that lets users view and compare bitmap files. A test bitmap can be compared to a baseline bitmap to evaluate application performance using multiple exclusion and inclusion regions.

Text Screen Viewer—The text screen viewer is a Presentation Manager application that lets users view and compare OS/2 VIO text screens. A test screen can be compared to a baseline screen to determine application performance using multiple exclusion and inclusion regions.

Power-Off and Power-On Support—The optional Omega Power Control feature allows ATF to recycle a machine's power after a catastrophic software failure, enhancing ATF's ability to run tests unattended. Even if this option is not exercised, ATF's architecture, with the Executive completely external to the test workstation, promotes unattended testing. On encountering a failed test, the Executive proceeds to the next script, loading and starting applications and test tapes as needed.

ATF Controller

An ATF Controller receives DTL commands from the Executive and runs the application to be tested on the Controller machine. Results of the tests are then sent from the Controller to the Executive.

Controller Features

Communication Between Executive and Test Applications—The Controller manages the interaction between the Executive, which orders specified tests to be performed with DTL, and the application being tested against which those tests are run. The presence of Controller software on a test workstation gives the Executive complete control over that machine.

GUI State Restoration—One of the most difficult problems in GUI test automation is

Now
only \$149.

The computer company that never takes shortcuts is delighted to present seven.

They're on the screens you see here. And they can end the most boring part of your job—writing code for GUIs.

Introducing CUA Controls Library/2™

It contains all the drudgery you'll never have to do again, if you're writing Windows™ or OS/2® applications on OS/2 1.3, Windows 3.0 or 3.1. It's all right here and it's all reusable. Over and over again.

With CUA Controls Library/2, you save time. You save money. And best of all, you can save your energy for the really creative parts of your work.

These screens are just a glimpse. From tables to graphics, from files to notebooks, from fonts to textures to tones, CUA Controls Library/2 does it all. To give you complete control over your program's functioning, organization, design and "look." All in a flash.

In a world as competitive as this, IBM definitely thinks you should take the easy way out. CUA Controls Library/2. For more information or to order, call 1 800 342-6672.

CUA Controls
Library/2

- Develop for OS/2 1.3 and Windows 3.0 and 3.1.
- Save time, money and resources.
- Call 1 800 342-6672 for more information or to order.

IBM®



The bottom line is that anyone building GUI networked applications cannot ensure software quality with only the brute force of manual testing.

restoring a test's beginning state for each instance of a test run. The ATF Controller provides sophisticated GUI state restoration capability. For example, before running a tape, the Controller ensures that an application's windows are of the same size and in the same position as they were when the original recording was made.

Satellite Tape Factory—In addition to driving a Controller from the ATF Executive, users can use a Controller as a satellite tape factory, permitting tapes to be recorded and played back without intervention from the Executive. These tapes, which may later be uploaded to the Executive for use in ATF scripts, are equivalent to tapes created directly through the Executive. The satellite user interface can also be used to start applications and snap bitmaps and window text.

As noted previously, ATF supports programmatic and tape record/playback approaches to testing. Although most of ATF's users follow a combined approach, a rule of thumb is that a capture/playback facility lets users get up and running with ATF very quickly. Users without any programming background can create a library of tapes that will become the basis for substantial test suites. A programmatic approach requires a higher level of programming ability, but DTL has proven easy to use. While it supports sophisticated programming through its range of commands and functions, in its simplest form, non-programmers can build straightforward test cases. Figure 2 illustrates a

simple ATF script that takes the programmatic approach, running the OS/2 editor application; Figure 3 is an ASCII version of a tape recording of the same user session.

While a tape may be easier to create (ATF's recorder does the work in terms of figuring out what is going on with an application), the result is more complex to edit. While simple, Figures 2 and 3 do illustrate the rudimentary ATF commands for connecting to the machine on which a test will occur, starting an application, playing keys, and making a menu selection.

The results of a test, whether done programmatically or via tape, are logged so testers can determine its success or the point of failure. The advantage of using scripts and tapes is that once a bug is encountered and a fix is made, a test can be rerun automatically, unlike with manual testing, which requires the tester to sit down and run through the application again.

Whether OS/2 developers are testing with ATF or not, the bottom line is that anyone building GUI networked applications cannot ensure software quality with only the brute force of manual testing. The testing future belongs to an increasingly automated approach to testing increasingly sophisticated applications.

Marvin Tener, Softbridge Microsystems Corp., 125 CambridgePark Dr., Cambridge, Mass., 02140, (617) 576-2257. Tener has over 25 years experience in software development and testing. Before assuming his current post as director of development for Softbridge, he was the development director of several major client/server systems implemented by Softbridge Consulting. He has also worked as an independent consultant and vice president of technology at Interactive Data Corp. Tener holds a B.A. in mathematics from the University of California at Berkeley.



WHY WASTE VALUABLE TIME...



**WHEN YOU CAN GO
STRAIGHT TO THE SOURCE?**



**Introducing the only magazine
exclusively devoted to software
developers working in the OS/2
environment.**

- ☐ **THE SOURCE** for OS/2 tips and techniques, direct from Big Blue to you.
- ☐ **THE SOURCE** for those working with everything from LANs to multimedia to DBMS architecture.
- ☐ **THE SOURCE** for valuable news and information for OS/2 application developers.
- ☐ **THE SOURCE** for new software development ideas and products.

Subscribe now and pay just \$39.95 for four big issues, delivered every single quarter to your office desk – straight from the source.

☐ **YES! Send me a full year of IBM OS/2 Developer. I'll pay just \$39.95 for four big issues!**

Name _____ Title _____

Company _____

Address _____

City/State/Zip _____

Charge My Credit Card : ☐ VISA ☐ MC ☐ AMEX

Card Number: _____

Expiration Date: _____ Signature: _____

You may pre-pay by enclosing a check or money order made out to IBM OS/2 Developer, or by enclosing VISA, MasterCard or American Express number, expiration date, and your signature. Delivery of first issue will be in 6-8 weeks. Canadian and international surface mail orders, add \$16 per year for postage. International surface mail is an additional \$30 per year for postage. Checks must be in U.S. funds.

SPEED YOUR ORDER!
1-800-WANT-0s2
1-708-647-5960
(OUTSIDE THE U.S.)

FAX:
1-708-647-0537
MAIL THIS FORM TO:
IBM OS/2 DEVELOPER
P.O. Box 1079
SKOKIE, IL, 60606

INSIDE INFORMATION

NETSEC'93 NETWORK SECURITY IN THE OPEN ENVIRONMENT

June 21-23 1993 • Capitol Hilton • Washington, D.C.

The dates of June 21-23 will be more important to you in the coming year than they've ever been before.

Because those are the dates of NETSEC '93: the world's only conference and exhibition dedicated 100% to network security in the business environment. Its exclusive focus on networks provides inside information on security that spans the spectrum – from administrative and management issues, to access and virus control, to physical security.

59+ REASONS TO ATTEND

In addition to the 50-plus workshops, seminars and case studies focused on network security, there are nine more reasons to attend NETSEC '93:

1. **YOU'LL GET** three full days completely dedicated to showing you how to secure your computer and telecommunications networks.
2. **YOU'LL LEARN** from some of the most experienced computer and information security practitioners ever assembled to examine the issues and solutions for securing networks in the 90s.
3. **YOU'LL FIND** out how your peers solve the problems inherent in securing today's information infrastructure.
4. **YOU'LL KEEP** up with the issues raised by new office automation technologies, such as FAX, e-mail, EDI, multimedia and more.

5. **YOU'LL DISCOVER** how to face the complex security management problems resulting from today's decentralization of computer resources.

6. **YOU'LL SEE** the latest network security products at a selective vendor exhibition designed to give you time to explore, examine and make informed purchasing decisions.

7. **YOU'LL LEARN** precisely what you want to learn about network security, by developing a customized learning experience based on the conference sessions of your choice.

8. **YOU'LL GET** a special discount on the most highly rated computer security training when you elect to attend the optional two-day CSI seminars held in conjunction with NETSEC '93 – a sure way to maximize your conference travel dollars.

9. **YOU'LL GET** immediate access to other privileged discounts, as well as the 1993 Computer Security Product Buyers Guide, when you decide to become a CSI member at the conference.

CHOOSE FROM 50+ SESSIONS SUCH AS:

- Network security policies and procedures
- Privacy throughout a network
- Managing the Standard Network Protocol
- Wireless LANs
- Capturing and prosecuting telecommunications fraud
- EDI: securing the business link
- Network security in a multi-vendor environment
- Risk analysis for networks
- Five-step check up for LANs
- Virus management on the network
- OS/2 LAN security
- AppleTalk network and Macintosh workstation security
- and many more

REMEMBER: The word is out. NETSEC '93 is where you should be June 21-23. Use the coupon to register or to get the details on this inside-information event for securing networks in the new open environment.

- ☐ **YES, I will attend NETSEC '93. Please register me.**
- ☐ I'm not ready to register yet, but please send me a conference catalog.

NAME	TITLE		
COMPANY			
ADDRESS	MAIL STOP		
CITY	STATE	ZIP	
PHONE	FAX		

NETSEC '93

JUNE 21-23, 1993
CAPITOL HILTON
WASHINGTON, D.C.
FAX TO:
(415) 905-2218
MAIL TO:
CSI CONFERENCE
REGISTRATION
600 HARRISON ST.
SAN FRANCISCO
CA 94107
OR CALL:
(415) 905-2626

Multimedia

OS/2 2.0 Video Systems: Structure and Behavior



by Weldon Adair

OS/2 2.0 can run 16- and 32-bit OS/2 applications as well as Windows and DOS applications, in either windows on the desktop or full-screen sessions. Applications run differently in each format, and each format presents its own advantages. Some full screen DOS and Windows applications, for example, may run at different resolutions under certain circumstances than they do on the Presentation Manager desktop.

This article explores the OS/2 session architecture and how it is supported by the OS/2 display drivers. It also covers the conditions under which programs may become suspended due to video operations and how to run full-screen Windows applications when the desktop is in medium resolution.

RUNNING DOS AND WINDOWS APPLICATIONS ON OS/2

OS/2 has always been able to run multiple OS/2 applications. Under OS/2 2.0, users can also run multiple DOS and Windows applications, most of them in windows on the OS/2 Desktop. This increased flexibility results in part from OS/2 support of the 386 architecture processor's virtual 8086 (v86) mode in a 32-bit paged address space. The v86 feature is used to construct virtual DOS machines (VDMs) that allow DOS applications to coexist with the protected-mode applications of OS/2. Using virtual device drivers (VDDs), which provide the resources of a virtual PC to the DOS program, a VDM runs in its own session like any other OS/2 process.

Only the display physical and virtual device drivers will be discussed in this article. For a fuller understanding of how OS/2 supports

VDMs, see *The Design of OS/2* (Deitel and Kogan, 1992).

DOS and Windows Applications in Full Screen

VDDs for display handle the output of DOS applications to the actual display hardware. Windows output is handled both by the VDD and by cooperation between the OS/2 physical device driver (PDD) and the Windows device driver. Understanding how the VDD provides support to an application running in full-screen mode helps us understand how the VDD supports output of a DOS application to a window.

DOS or Windows applications running in full-screen mode in the foreground fill the entire screen, obscuring the view of the OS/2 desktop. Only one full-screen application can access the display at any one time; only one application, therefore, is actually accessing the display hardware.

Under most circumstances, the VDD provides a virtual display to the application running in the VDM. But with an application in full-screen mode, the actual display receives exactly the same output generated by the application. In this situation, the VDD can be thought to have turned the display hardware over to the DOS application or Windows device driver.

But if the VDD truly turned over all display hardware, a DOS application or Windows device driver could put the display into a state unknown to and unsupported by the OS/2 PDD. To protect the system, the VDD must capture the state of the display hardware before "surrendering" it to uncontrolled conditions.



Weldon Adair

VDM: Virtual DOS Machine

VDD: Virtual Device Driver

PDD: Physical Device Driver



In actuality, a VDD never completely surrenders the hardware to an application running in a VDM. Under certain circumstances, the VDD may allow the application some direct access to the hardware; however, the VDD always returns the display to the OS/2 desktop in the condition expected. Usually, the VDD intercepts all VDM attempts to directly access display hardware and BIOS calls. In some cases, the VDD saves information and updates the display hardware as an application is given focus. At other times, the VDD simply updates the display hardware.

Upon returning control to the desktop, the VDD changes modes of operation. In its new mode, it continues to receive input from the VDM application but does not allow direct access to the hardware.

DOS and Windows Applications in Windows

Upon returning control to the desktop, the VDD changes modes of operation.

When a DOS application is run in a window on the PM Desktop, the VDD intercepts all VDM application attempts to output directly to the display hardware. In this case, most of the hardware function is emulated by the system and then passed to the PDD. Windows applications run in a window on the Desktop, in contrast, make use of cooperation between the PDD and the special "seamless" Windows (WIN-OS/2) device driver.

There are limitations on DOS graphics applications run in Windows on the PM Desktop. DOS graphics applications must not operate at a higher video resolution than that of the PM Desktop, and software cannot emulate a higher-resolution mode than that of the display. If a DOS graphics application attempts to set an unsupported mode, the DOS VDM is suspended.

If a DOS graphics application manipulates the display palette in full screen mode, it may not have the desired appearance in a window. While the VDD maps the DOS application colors to the PM Desktop palette with the closest fit possible, there may be slight color variations because of the transfer. If a DOS graphics application uses the mouse to control the cursor, you may have to change the DOS settings `MOUSE_EXCLUSIVE_ACCESS`. DOS text applications can be in any of the supported

text video modes; text modes are emulated in graphics mode when run in a window.

In a window on the PM Desktop, Windows applications run very differently than DOS applications. The VDD for Windows in a PM window (VWIN.SYS) is slightly different from the DOS VDD, providing limited addressability to a small set of PM routines and data and hardware serialization services. The special "seamless" WIN-OS/2™ device driver and the OS/2 PDD are aware of each other, coordinating their activities through a semaphore and sharing access to the display hardware. The WIN-OS/2 display driver actually "owns" a piece of the display. Because this mode of operation requires that the WIN-OS/2 device driver and the OS/2 PDD cooperate, they must be used in pairs. Older OS/2 and Windows display drivers that have not been modified to coordinate their activities cannot run WIN-OS/2 in a window.

BACKGROUND EXECUTION OF GRAPHICAL APPLICATIONS

When a DOS application is switched from full-screen foreground to background operation, it can no longer have the direct access to display hardware it enjoyed in the foreground. In this case, the VDD restores the hardware to the state it was in just before the DOS application was given control of the full screen. The VDD also begins to intercept all the DOS application's attempts to access the display hardware, using the intercepted information to update a virtual display. Some operations that can be done by a DOS application in a full-screen session are not emulated; if the DOS application attempts one of these operations, the VDM is suspended.

When a DOS application run in a window on the PM Desktop is switched to the background, no great changes are made, as the VDD is already intercepting the application's attempts to access the hardware. The biggest change in this situation is that part of the virtual display is no longer displayed on the PM Desktop.

When a WIN-OS/2 session is switched from full-screen foreground to background operation, display drivers provided for WIN-

OS/2 full-screen sessions suppress all background write operations but allow the WIN-OS/2 program to continue to run in the background. If the session is switched to the foreground again, the program is prompted to redraw. When a WIN-OS/2 application running in a window is switched to the background, the seamless WIN-OS/2 device drivers suppress writing to the display in a manner similar to that of the full-screen device drivers.

Windows device drivers not provided for WIN-OS/2 full-screen sessions may not suppress background writing to the display. If they do not, the WIN-OS/2 session may be suspended in the background.

High-Resolution WIN-OS/2 in Full-Screen Mode

In the initial release of OS/2 2.0, WIN-OS/2 window sessions are supported in VGA mode only. For Windows users who depend on higher-resolution applications, this limitation may seem unacceptable. However, there is a way to run WIN-OS/2 windows sessions and PM Desktop (Workplace Shell) in VGA mode, while running higher-resolution Windows applications in full screen mode using the SYSTEM.INI and WIN.INI files.

In the SYSTEM.INI file (found in the OS2\MDOS\WINOS2 directory), there is an entry specific to WIN-OS2 windows sessions, sdisplay.drv, used to designate the WIN-OS2 windows session video device driver. (The "s" in its name stands for "seamless.")

The SYSTEM.INI file also contains the display.drv entry from Windows. At OS/2 2.0 release time, the swinvga.drv was the only WIN-OS2 windows session video device driver available from IBM.

As you might guess, display.drv designates the use of the full screen WIN-OS2 video device driver. If display.drv points to a high resolution device driver that supports installed hardware, full screen WIN-OS2 will run using that driver. This configuration provides the best of high-resolution Windows and seamless WIN-OS2.

The following section presents a detailed step by step procedure for using this feature. This

section assumes that the reader is not familiar with the SYSTEM.INI and WIN.INI files.

These instructions should be followed very carefully. If caution is not observed, it is possible to render inoperable seamless and full screen WIN-OS2 sessions.

The first step is to install OS/2 for medium resolution (VGA). After this is done, it is very important to make two backup copies of the SYSTEM.INI and WIN.INI files, one to a floppy disk or tape and the other to the OS2\MDOS\WINOS2 directory.

```
[C:\]dir a:*xga*
The volume label in drive A is DISK nn.
Directory of A:\
SYS0002: The system cannot find the file specified.
[C:\]dir a:*G.FON
The volume label in drive A is DISK nn.
Directory of A:\
SYMBOLG FON 24532 3-17-92 5:35p
          1 file(s) 24532 bytes used
          130560 bytes free
```

Figure 1: Search for FON files

It is assumed that a disk or tape backup copy have been made; following are instructions for backing up to the directory.

- Back up the SYSTEM.INI and WIN.INI files.
- Open an OS/2 window from the Command Prompts folder in the OS/2 System folder. At the [C:\] prompt, input:

```
[C:\]cd\os2\mdos\winos2
[C:\os2\mdos\winos2]copy win.ini win.bak
          1 file(s) copied.
[C:\os2\mdos\winos2]copy system.ini
system.bak
          1 file(s) copied.
```

- Once the SYSTEM.INI and WIN.INI files are safely backed up, proceed with modifications.

Before you make the changes to the files, it is best to make sure all desired high-resolution Windows device drivers and fonts are loaded on the system.

Check the \os2\mdos\winos2\system directory for the desired high resolution display device





```
[C:\]unpack a:symbolg.fon
a:SYMBOLG.FON
- \OS2\MDOS\WINOS2\SYSTEM\SYMBOLG.FON
  0 file(s) copied.
  1 file(s) unpacked.

SYMBOLG.FON
TMSRG.FON
COURG.FON
HELVG.FON
```

Figure 2: Unpack FON files

```
[C:\]unpack a:winxga
a:WINXGA
- \OS2\MDOS\WINOS2\SYSTEM\XGA.DRV
- \OS2\MDOS\WINOS2\SYSTEM\XGASYS.FON
- \OS2\MDOS\WINOS2\SYSTEM\XGADEM.FON
- \OS2\MDOS\WINOS2\SYSTEM\XGAFIX.FON
  0 file(s) copied.
  4 file(s) unpacked.
```

Figure 3: Unpack driver

```
fixedfor.fon=vgafix.fon
oemfonts.fon=vgaoem.fon
fonts.fon=vgasys.fon
```

For XGA, these entries should be changed to:

```
fixedfor.fon=xgafix.fon
oemfonts.fon=xgaoem.fon
fonts.fon=xgasys.fon
```

In WIN.INI, these entries will need to be changed:

```
Symbol 8,10,12,14,18,24 (VGA res)=SYMBOLG.FON
Helv 8,10,12,14,18,24 (VGA res)=HELVG.FON
Tms Rmn 8,10,12,14,18,24 (VGA res)=TMSRG.FON
Courier 10,12,15 (VGA res)=COURG.FON
```

Change this to:

```
Symbol 8,10,12,14,18,24 (XGA res)=SYMBOLG.FON
Helv 8,10,12,14,18,24 (XGA res)=HELVG.FON
Tms Rmn 8,10,12,14,18,24 (XGA res)=TMSRG.FON
Courier 10,12,15 (XGA res)=COURG.FON
```

Figure 4: Modify WIN.INI file

driver. This example uses the XGA.DRV device driver. Checking the directory reveals that this driver is not loaded. (If the system was installed in high resolution mode and had a selective install to return to medium resolution mode, the driver may be loaded already. A description of this approach is given below.)

Retrieving the device driver and fonts from the installation disks may involve a bit of hunting. Because the OS/2 build system is packed to optimize disk space and use as few disks as possible, the files may be found on any disk from number six to number 15.

For XGA, search the disks for *XGA* and *G.FON, as shown in Figure 1.

In the example in Figure 1, no XGA drivers were found on the disk, but one font file was. Although the font file is not part of a bundle of files, it still must be unpacked.

As the files are packed with their standard target directory coded into the packed file, running the unpack utility will copy the file to the proper directory on the system disk, as in Figure 2. Eventually the WINXGA file, containing the WIN-OS2 XGA.DRV and three XGA fonts, is found using the `dir a:*XGA` search, as in Figure 3.

With all required driver and font files on the system disks, the next step is to modify the SYSTEM.INI file. Edit the SYSTEM.INI file and find the line:

```
display.drv=vga.drv
```

This line tells WIN-OS2 which device driver to use in full screen sessions. Change this line to point to the high-resolution device driver unloaded in the previous steps. In this case, the device driver is XGA.DRV. The modified line should look like:

```
display.drv=xga.drv
```

Font entries in both SYSTEM.INI and WIN.INI should be changed. Figure 4 lists the entries for SYSTEM.INI.

Once these changes have been made and the files saved, the system is ready to run seamless WIN-OS2 applications in VGA mode and full-screen WIN-OS2 applications in high-resolution mode.

Weldon Adair, Computer Sciences Corp., 16511 Space Center Blvd., Houston, Texas 77058. Adair is a senior computer scientist with Computer Sciences Corp. (CSC). He holds degrees in physics and mechanical engineering and has over 20 years experience with computers. He is currently working for CSC on a contract for NASA in Houston, Texas. During the development of OS/2 2.0, he worked in system development at the IBM Boca Raton Programming Center.

REFERENCES

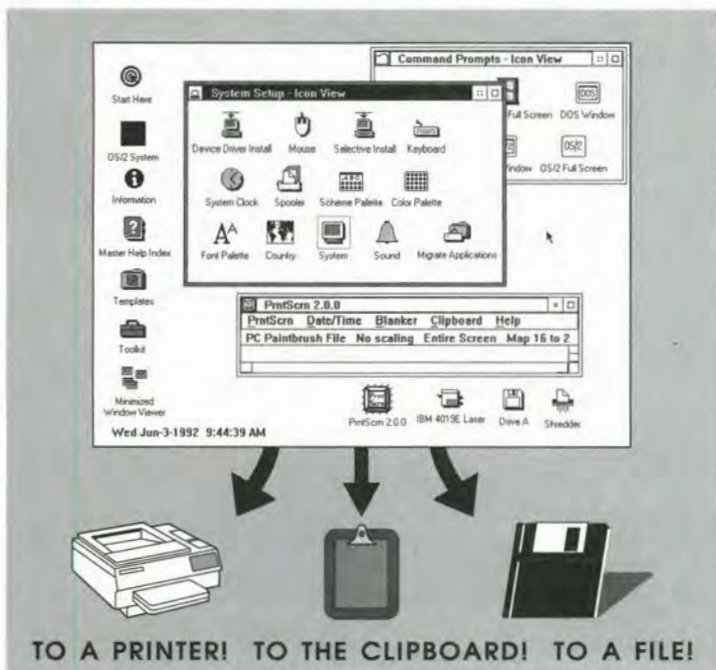
Deitel, H.M. and M.S.Kogan. *The Design of OS/2*. Reading, Mass: Addison-Wesley Inc., 1992. (IBM Doc. 5325-4005)

For a full description of background execution of DOS and Windows graphical application programs, see the *OS/2 2.0 Compatibility Information Booklet* (IBM Doc. 41G8276), included in the documentation shipped with OS/2 2.0.



CAPTURE YOUR SCREEN

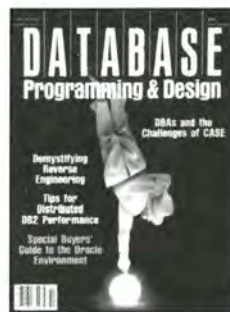
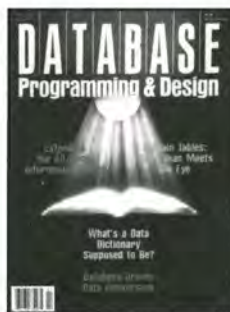
PrntScrn™ is THE One-Step Screen Image Utility for copying a graphics image from the Workplace Shell screen to a LAN or locally attached printer, to the Clipboard, or to a disk file. The screen images appearing in this publication were captured and processed using the **PrntScrn** utility. The capture operation can be initiated from the keyboard (by pressing the print-screen key), from the mouse, or from another program. The captured image area can be the entire screen, current active window, current active client area, selected window, selected client area, or specified rectangular area. Disk file formats include PM Metafile, PM Bit Map, PCX, TIFF, GIF, and WPG. The "Color Mapping" feature provides the control needed to produce quality printed images from color screen images. **PrntScrn** has clipboard Viewing, Printing, Exporting, & Importing capabilities for Bit Maps, Metafiles, and Text files. **PrntScrn** includes a Screen Blanker and a digital Date & Time "information bar". **PrntScrn** is priced at \$115 and is available from major software resellers, or contact MITNOR Software.



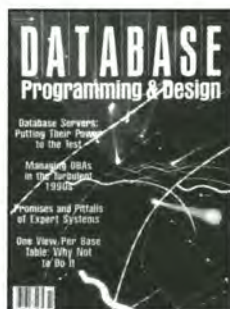
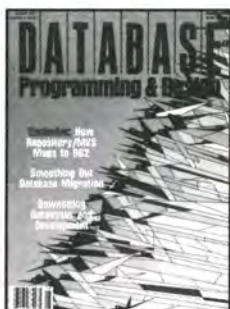
MITNOR Software
28411 E. 55th Street
Broken Arrow, OK 74014

Phone: (918) 357-1628
Fax: (918) 357-2869





WE WOULD LIKE TO SURROUND YOU WITH SOLUTIONS.



Information at your fingertips.

It sounds like a cliché. But when you have a good magazine in your hands the cliché becomes a reality.

Every issue of Database Programming & Design puts information and solutions at your fingertips.

Solutions that will increase your database performance, reduce backlog and speed applications development.

Solutions that will, yes, make your job easier.

So no matter what your database specialty, you'll find a subscription to Database Programming & Design to be a real bargain. And at twenty percent off the regular subscription price, this bargain is now a real steal.

DATABASE Programming & Design



Solutions!

How can I possibly lose? If I become dissatisfied with Database Programming & Design at any time, I may cancel my subscription and receive a *full refund* of my \$37.00 investment.

Check one:

- ☐ Payment enclosed.
☐ Bill me.

☐ Yes! Please start my full-year subscription to Database Programming & Design for \$37.00—a 21% savings.

Name _____ Title _____

Company _____

Address _____ City _____

State/Province _____ Zip/Postal Code _____

Note: Subscription rate is for US delivery only. Canadian delivery: US\$43.00. International delivery: US\$52.00 for surface mail; US\$77.00 for airmail. Please send payment with order in US funds drawn on a US bank. Please allow 6-8 weeks for delivery. CA residents add \$1.45 state tax.

3 Easy Ways To Subscribe!

☎ BY PHONE: 1-800-289-0169

✉ BY FAX: 1-415-905-2233

✉ BY MAIL:
Database Programming & Design
P.O. Box 53481
Boulder, CO 80322-3481

International

OS/2: The International Scene



by Vicky Gallop

EMEA DEVELOPER ASSISTANCE PROGRAM

The EMEA DAP has expanded to provide marketing support to all its members in Europe, the Gulf countries, and Africa. Marketing support was announced to members in September, and was operating by early October 1992. Member response has been enthusiastic, with major developers such as Lotus and Borland International participating. Several new support programs are available:

- OS/2 Award
- OS/2 Compatibility Logos
- OS/2 Application Product Publicity
- OS/2 Joint Developer Events
- OS/2 Customer References
- OS/2 Solutions Book.

Details of these programs are available on the DAP online network. Companies wishing to join the DAP should reach their country contact, listed on the following pages.

EMEA DAP membership, recruited through these country contacts, totalled well over 1,000 by the end of 1992 and is increasing rapidly.

WORLDWIDE DAP

Direct on-line technical support and other OS/2 information is now available to developers around the world over

CompuServe. Log on to CompuServe and enter GOOS2DAP to enroll.

VENDOR SUPPORT IN SWITZERLAND

National developer support programs are growing in scope and activity. This issue, we focus on the ISV (independent software vendors) Support Program in Switzerland. All Swiss developers can participate in the program; they need only fill out a questionnaire. (Approval is given by Max Merz; participation is restricted to vendors without other IBM contacts.)

All participants have access to ISV relation contact Max Merz, who can help with development questions about hardware, software, developing, customizing, documentation, education, and contact with other IBM departments. Hardware and software announcements and information on education and strategies are distributed continuously. Members also have access to twice-yearly ISV seminars with national and international speakers.

DIAL IBM access is provided on request, as is authorization for solution associates, PS consultants, or remarketer sales points, if developers meet special conditions. In addition, the IBM customer support center facilities are available to independent vendors for customer seminars.

Special Support Offering

OS/2 application developers can buy complete systems (hardware and software) at a large discount. This offering is based on a contract in



Vicky Gallop

*Log on to
CompuServe
and enter
GOOS2DAP
to enroll in
the DAP.*

OS/2 COUNTRY CONTACTS

Australia	Rohaini Cain Sydney	61-2-354-7684
Austria	Georg Haschek Vienna	43-222-21145 x2335
Belgium	Luc Gheysens Brussels	32-2-025-3662
Brazil	Wagner Okutani de Sumare Almeida	55-192-65-7138
Canada	Ken Faddelle Toronto	1-416-946-3786
Czechoslovakia	Stepan Hradecny	422 7106111
Croatia	Zoran Kezman	3041 624500
Denmark	Poul Hansnes Copenhagen	45-45-93-4545 x4522
Egypt	Shamel Abaza Cairo	20-2-349-2533 x269
Finland	Ilkka Ayravainen Helsinki	358-0-4594007 x4004
France	Jean-Paul Rambaud Paris	33-1-48154660
Germany	Hans-Michael Oest Stuttgart	49-711-785-2417
Greece	Chryssina Hadjitheodorou Athens	30-1-32-81111
Gulf Countries	Robert Kikano Bahrain	973-210880
Hong Kong and Peoples' Repub. of China	Roland Stranneborn	852-825-6996
Hungary	Stevan Szarka	361 1654422 x108
China	Roland Stranneborn	852-825-6996
Iceland	Poul Hansnes (covered by Denmark)	45-45-93-4545
India	Tang Wek Soon (covered by Singapore)	65-320-1181
Ireland	Barry O'Brien Dublin	353-1-603744 x4629

which the developer specifies the new program and the planned release date.

OS/2 APPLICATIONS AROUND THE WORLD: SWEDEN***ABB InfoSystems***

Sweden's ABB Infosystems develops the SIGMA™ system integration manager, which integrates applications and application packages running in different computer environments, without any adaptations or changes in the applications. It became available in late 1992.

ABB InfoSystems is a full-service information technology company with approximately 50 employees. A member of the ABB Group, ABB InfoSystems has a turnover of approximately 120 million SEK (Swedish Kronen). The company has products and services in several areas: information systems strategies, system development, and IS operations and systems management.

The SIGMA Project

The SIGMA project at ABB InfoSystems was spurred by customers' need for integration between different computer environments such as the PS/2, AS/400™, and ES/390™. IBM provided OS/2 support and network design for SIGMA, and ABB InfoSystems used IBM workstation technology including OS/2 2.0, Extended Services, LAN Server, the Developer's Toolkit, and the C SET/2 compiler.

Designed for companies that want to integrate standard packages, tools, and in-house applications, SIGMA allows users to approach corporate EDP systems (made up of independent subsystems) as one unified system. It can mix applications and tools running on different hardware and software platforms.

With SIGMA, users can reach required systems through the menu or the icons available on a PC, bypassing the system's menu and icons. A menu or icon selection can be created to perform a number of processes in one or several applications. SIGMA can also exchange information between applications and tools, restructuring data when necessary or performing a logging function for audit routines. A report-ordering function that creates ordered reports using different systems' report generation functions (the agent technique or desired selection) can be ordered with only one menu selection. SIGMA can also review or reprint old reports, exchange a tool or an application, or create new information exchange links between subsystems.

All the previous facilities are controlled by the SIGMA Engine™ software running under OS/2. The SIGMA Engine is the domain controller over all DOS- and LAN-based applications integrated through SIGMA. All workstations using SIGMA must be connected to the SIGMA Engine by an IBM LAN Server running Extended Services.

SIGMA supports dialogues and applications operated from SIGMA workstations and manipulated in OS/2, PC DOS and MS DOS, or the IBM 5250 and 3270 terminals.

ACKNOWLEDGMENTS

Information on the Swiss ISV program was provided by Swiss DAP contact Max Merz.

For more information on ABB Infosystems, contact Rune Hultkvist in Sweden at +46 36 171218.

Vicky Gallop, IBM U.K. Ltd., Mountbatten House, Basing View, Basingstoke, Hants., RG21 1EJ, U.K. Gallop is a marketing specialist in the OS/2 Group at the EMEA Personal Systems Centre. She worked as a statistician and computer consultant before joining IBM. Gallop holds a B.Soc.Sc. in economics from the University of Birmingham.

OS/2 COUNTRY CONTACTS

Israel	Arnan Dar Tel Aviv	972-3-697-8111
Italy	Giuseppe Cigala Milan	39-2-5962.5178
Indonesia	Zakaria Ansori Jakarta	62-21-520-9500
Japan	W. Kumada Tokyo	81-3-3808-4290
Korea	Bong Hun Park Seoul	82-2-781-6114
Malaysia	Peter Lim Kuala	60-37177788
Mexico	Juan Carlos Fernandez Sarda Mexico	52-5-557-8588 x1846
Netherlands	Hans Langenhorst Amsterdam	31-20-5653983
New Zealand	Rohaini Cain Sydney	61-2-354-7684
Norway	Kjell Tornby Oslo	47-2-99-93-83
Pakistan	Inayat Ali Ebrahim Karachi	92-21-525181-190
Philippines	Mike Valdes Manila	632-819-2000 x348
Poland	Alexander Siatecki	482 6582991
Portugal	Jose Carlos Matos Alves Lisbon	351-1-7955161
Romania	Constantin Florea	401 614 3929
Saudi Arabia	Firas Halawani Jeddah	966-2-6600007
Singapore	Lisa Goh Singaopre	65-320-1181
South Africa	Anita Volker Johannesburg	27-11-224-9111
Spain	Ignacio Martin Ibanez Madrid	34-1-397-9231
Slovenia	Dejan Podgorsek	3061 441102

OS/2 country contacts (continued on page 72)

OS/2 COUNTRY CONTACTS

Sweden	Roger Labrell Stockholm	46-8-793-1000 x1697
Switzerland	Max Merz Zurich	41-1-207-3377
Thailand	Pontip Naruenartwanich Bangkok	66-2-273-4267
Taiwan	Steve Kang Taipei	886-2-776-7530
Turkey	Sevgi Gurbuz Istanbul	90-1-280-0900 x1137
U.K.	Nicola Colborne Basingstoke	44-256-343095
Worldwide	Judy Osborne Boca Raton DAP	1-407-982-4259

OS/2 country contacts (continued from page 71)

**TO ORDER OS/2 DEVELOPER
OUTSIDE THE U.S.**

The OS/2 User Group
Barton House
Cirencester, Gloucestershire
GL7 2EE, U.K.
Tel: (0285) 641 175
Fax: (0285) 640 181

Cost is £40 U.K. for OS/2 User Group or DAP members, £50 UK for nonmembers, plus £8 for postage and packing. The magazine can also be ordered through the OS/2 country contacts (see sidebar).

The *OS/2 Applications Solutions Directory* is also available for subscription outside the U.S. for £19.95 U.K. plus postage (£23 airmail, £6.75 surface) and a £15 bank charge. The *Directory* may also be ordered through SLSS (IBM Doc. G362-0002).

**NDP Fortran, C/C++
and Pascal Compilers**

- VMS, VS and MS extensions
- Compatible with Framework and C Set — Fortran can call C Set or NDP C
- Uses IBM's Link386 to generate native OS/2 executables
- Can directly call the OS/2 APIs
- Generate globally optimized 32-bit mainframe quality code
- Support for all coprocessors and the i860
- Available for DOS, OS/2, UNIX, NT and Coherent
- 3rd party numerics and GUI Class libraries available

For more information, call our Tech Support Group
at (508) 746-7341.

Microway

Research Park
P.O. Box 79
Kingston, MA 02364

**Announcing TLIBTM 5.0!
Version Control for DOS & OS/2**

• The experts loved TLIB 4:

"...amazingly fast... TLIB is a great system." **PC Tech Journal**
"TLIB has features and power to spare... TLIB is easy to use and the fastest of the reviewed packages." **Computer Language**
"I will not program without it." **Uptime Magazine**

• Now TLIB 5.0 adds:

Automatic branching. Automatic version labeling across branches. Language-independent preprocessor, for "conditional compilation" in languages which do not support it (like dBase). N-way-tree version numbers. Branch and whole-library locking. OS/2 support.

• Plus the features they loved in TLIB 4:

Check-in/out locking. Branching, for parallel development. Keywords. Full binary file support (does not depend upon CRs in the file like other products). Wildcard and list-of-file support; can create lists by scanning source code for includes. Can merge (reconcile) multiple simultaneous changes and undo intermediate revisions. Network and WORM optical disk support. Mainframe-compatible delta generator for Pansophic, ADR, IBM, Sperry formats. Includes integrated PD MAKE by L. Dyer; also integrates with OpusTM MAKE, SlickTM MAKE, others.

MS-DOS \$139, OS/2 (with MS-DOS) \$195 + shipping.
5 station network: MS-DOS \$419, OS/2 \$595. Call for other sizes.

**Burton Systems Software**

PO Box 4156, Cary, NC 27519 (919) 233-8128

A Closer Look Is Very Revealing!

OS/2 Accreditation
Opens Doors *and*
Closes Deals

Certificate of Accreditation I.T. Expert

Has Successfully Completed
the OS/2 Examination
For Subject Matter Experts.
Valid For One Year From
January 1, 1993

Wilde
IBM Corporation

The
I.V.
League™

A directory of Independent Vendors
providing products and services that
support IBM OS/2®

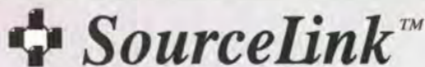


The "OS/2 Accredited" logo signifies your expert status as a trainer, consultant or system integrator of OS/2 2.X. Successful completion of the OS/2 Accreditation Examination allows you to display the logo on your marketing materials, stationery and business cards. To order the examination, remit \$149.00 for each individual taking the exam, in check or money order to: Skyline Consulting 246 Wolcott Road, Suite 158, Wolcott, CT 06716

Or for more information call (203) 879-6486

For information on products and services from OS/2 Independent Vendors, fax 914-766-3788 for a free directory.

Find Code **Fast** • Change Code **Fast**
Learn Code **Fast** • Navigate Code **Fast**



SourceLink is an OS/2 programming development tool combining the speed and power of hyper-link source code access with a fully functional editor. The edit screens are context sensitive. Click to access on-line help. Click-click to access your source code. With over 100 menu items to choose from, SourceLink is a feature rich programming environment.

Double click on a function, symbol or other string. Let SourceLink pop up a list of occurrences. Double click on each occurrence and SourceLink will immediately display the source code. Use the integrated editor to change or create new code.

Unmatched in speed and convenience for finding and changing source code. Ideal for porting code, changing code, analyzing code, and creating programs from existing code, templates and samples.

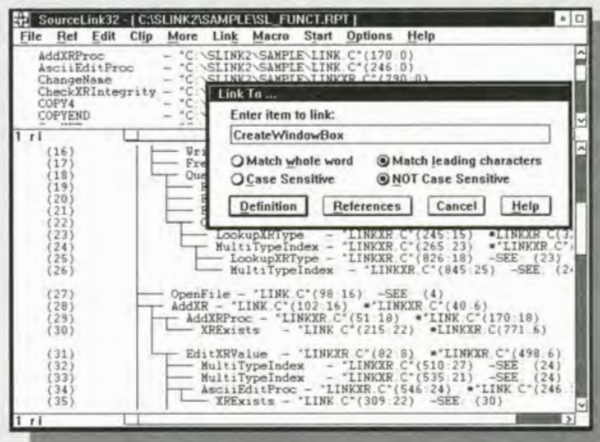
SourceLink will save you valuable programming time and energy! SourceLink is a refreshing new experience in programming!

Features:

- HyperLink source code access.
- Fully functional editor.
- Supports 'C', C++, MASM.
- REXX macro language interface.
- Presentation Manager GUI.
- WorkFrame/2 compatible.
- Generates function call tree displays.
- Context sensitive edit windows.
- Many file manipulation utilities.
- Find Files, Delete, Copy Files/ Directories.
- Context sensitive View Help.
- Multi-File/Dir. string search.
- Project configurable.
- Spawn compiles and commands.
- Create your own code templates.
- Multiple windows.
- Cross reference globals, symbols, dialogs, API calls, and more.
- Also available for OS/2 1.3.
- An OS/2 2.0 32 bit application.

SourceLine Software, Inc. - 7770 Regents Rd #113-502 - San Diego, CA 92122 - (619) 587-4713

Your Satisfaction is Guaranteed



World's most powerful tools for OS/2®

Hamilton C shell™

The superior alternative to the standard OS/2 command processor. Faithfully recreates and updates the entire UNIX® C shell language. Created explicitly for OS/2 using modern incremental compiler technology. Not one line ported from or created on anything but OS/2. Very high performance. Extensive support for multi-threading.

Features: Full-screen command line editing • Filename and command completion • History • Arrow and function keys • Unlimited size command lines • Recursive filename wildcarding • Fully nestable control structures • Command substitution • Aliases and shell procedures • PATH hashing • Background threads and processes.

Over 130 commands: alias, cat, chmod, cls, cp, cut, diff, dirs, dskread, dskwrite, du, eval, fgrep, grep, hashstat, head, history, label, ls, kill, markexe, more, mv, popd, printf, ps, pushd, pwd, rm, sed, sleep, split, strings, tabs, tail, tar, tee, time, touch, tr, uniq, vol, wait, wc, whereis, xd and others.

Meticulously adheres to OS/2 conventions. Supports HPFS, long filenames and all networks under OS/2 1.2 or later and 32-bit and VDM applications under OS/2 2.0.

\$350.00. Unconditional satisfaction guarantee. (\$365 in Canada, \$395 elsewhere.)

Hamilton Laboratories

13 Old Farm Road, Wayland, MA 01778-3117
Phone 508-358-5715 • FAX 508-358-1113

Golden

CompuServe® Access Software

Get fast answers to your OS/2® development and usage questions!

Reduce the cost of communicating with IBM OS/2 technical support on the OS/2 Information Exchange!

Use CompuServe intuitively with a multi-threaded OS/2 application!

Only \$99

Satisfaction Guaranteed

**Creative Systems
Programming Corporation**

Phone: (609) 234-1500
Fax: (609) 234-1920
Internet: 71511.151@compuserve.com

\$15 CompuServe usage credit included



CommPass

Tools

AlertVIEW: Network Management For Applications and Operating Systems



by Shlomo Touboul

This article shows how to use Shany's AlertVIEW™ API to send a SNA Generic Alert from any DOS, Windows, or OS/2 application to the IBM LAN Network Manager and NetView™.

AlertVIEW was created by Shany and IBM to fill a gap in existing network management services. Although there are literally hundreds of management programs on the market, they are limited to the physical components of the network—hubs, cables, bridges, and so on.

The most vital function of the LAN, the reason it exists, is to run applications and communications software. If users cannot supervise and monitor these programs, they run virtually without control.

In a mainframe environment, application management is supplied by software systems such as NetView. With the proliferation of companies downsizing applications to LANs, most of this application management is lost. AlertVIEW, with its network management for applications and its Developer's Kit, is the first product to address this problem.

ALERTVIEW

The AlertVIEW product line enables users to monitor and supervise critical corporate applications from a single management console.

The AlertVIEW workstation's agent (AVS) is an "application sniffer" that looks into applications, locates the source of a problem, and sends SNA generic alerts to the IBM LAN Network Manager and IBM NetView. The AVS detects about 90% of application failures, which can be due to operating system errors,

network operating system failures, communication problems, hardware faults, and so on. The remaining 10% are application logic errors such as an application infinite loop. This can be reported with the AlertVIEW Developer's Kit.

AlertVIEW Developer's Kit

The AlertVIEW Developer's Kit is an easy to use application program interface (API) for the creation and transmission of IBM SNA alerts from any DOS, Windows, or OS/2 application to the IBM LAN Manager, LAN Network Manager (LNM), or NetView console. It is shown in Figure 1.



Shlomo Touboul

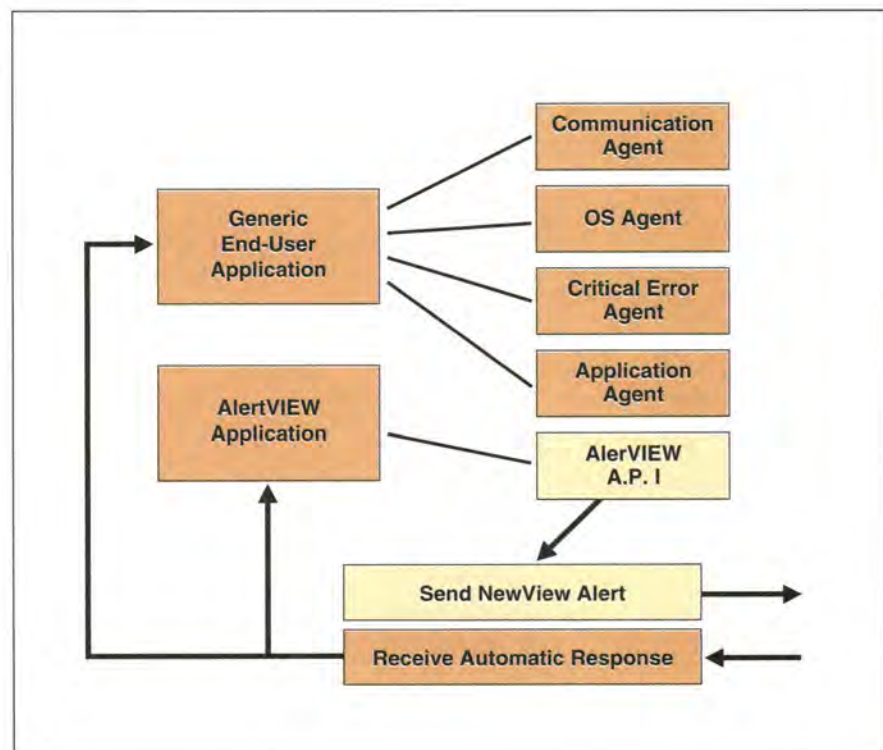


Figure 1: AlertVIEW Applications' Agent and API



Using the `SendAlert` system call interface, a programmer may send application-generic alerts to the IBM focal point. Developers can use the AlertVIEW API to send application-critical logical errors such as "illegal login password," "application out of resources," or "abnormal application termination," so applications can be continually supervised by NetView. This same API may be used for network management systems that are not compatible with IBM LAN Manager, LNM, or NetView.

IBM standard NMVT vector, and use the LLC 802.2 communication to deliver the alert to the closest entry point. Developers don't have to learn SNA formats, complex NMVT vector and sub-vector structures, or low-level network management protocols (LLC 802.2). With the AlertVIEW API, applications benefit from an alert service that until now was only available for mainframe applications. Because it supports additional network management protocols such as SNMP over IPX, AlertVIEW will support a multi-NMS protocol environment.

API CALLS

From C:

```
/* call to AlertVIEW API */
err_code=SEND_ALE ("Cannot write to file MYDATA.DAT,
please call your LAN Admin" );
If (err_code) /* check error code */ print f ("AVAPI
ERROR %d\n",err_code); /* print error message */
```

From Turbo Pascal:

```
{ Call to AlertVIEW API }
my_aps := 'Cannot write to the file MYDATA.DAT, please
call your LAN Admin';
err_code :=SEND_ALE (my_aps); { AVAPI system call }
if err_code<>0 then { check error code } writeln ('AVAPI
ERROR ',err_code); { print error message }
```

From assembler:

```
;AVAPI Parameters String
my_aps DB "Cannot write to file MYDATA.DAT, please call
your LAN Admin", 0
mov ax, SEG my_aps mov ds, ax ; APS segment mov dx,
OFFSET my_aps ; APS offset mov ax, 1 ; function code int
0f0h ; call AVAPI jc AVAPI_error ; jump if error
AVAPI_error:
```

Figure 2: API calls

The API consists of a single input parameter made up of the `AlertString` and a call from any PC programming language to the `SendAlert` procedure. The API will perform all functions necessary to transform the text message to an

Alert Control

While you can use the AlertVIEW API by simply sending an alert string to the API system call, you can specify and control all the IBM code points (text indexes that are transformed to text on the IBM LAN Manager, LNM, or NetView console). If no code points are specified, the AlertVIEW API sets the default code point for software error, using special string characters in a way similar to the C `printf()` procedure. You may set up each code point to specify the alert's probable cause, recommended action, user cause, and so on.

AlertVIEW API Implementation

For DOS workstations, the API run-time application is a DOS terminate-and-stay resident (TSR) program requiring 3K. The TSR extends the DOS services and supports the `SendAlert` system call using 80x86 interrupts similar to those of the DOS system calls.

For MS-Windows workstations, the API run-time application is a Windows dynamic link library that can be called from any Windows 3.0 or 3.1 application. The dynamic link library communicates with the DOS API TSR interface using the DPMI interface.

For OS/2 workstations, the API run-time application is an OS/2 dynamic link library that can be called from any Presentation Manager or OS/2 text application.

Using API from a command-line interface, a `SENDALERT.EXE` program for the DOS, Windows, and OS/2 platforms is also available to send alerts from the command-line interpreter or any batch file.

Supported Programming Languages

The Developer's Kit includes a built-in support library for the following programming languages:

- PC ASM86 • Microsoft COBOL
- Turbo Pascal • Microsoft PASCAL
- Borland C • Microsoft BASIC
- Microsoft C 5.1 and 6.0

EXAMPLES FOR API CALLS

Examples of API calls from C, Turbo Pascal, and an assembler language are shown in Figure 2.

ALERTVIEW API OPERATION

The AlertVIEW API module may reside on any LAN station, as shown in Figure 2. Its main

task is to interact with any application running on the LAN station that needs to send alerts to NetView to report user-defined alerts.

When the API calls take place, the AlertVIEW API module uses an internal interpreter that generates an NMVT vector from the programmer's alert string. The NMVT is transmitted over an 802.2 DLC interface or Novell IPX interface to the AlertVIEW Event Monitor, IBM LAN Manager, and IBM LAN Network Manager.

From the perspective of the entry points, there is no difference between an alert generated by the AlertVIEW API (or by a user application) and alerts generated by any other SNA components such as the 8230.

ALERTVIEW PRODUCT LINE

The AlertVIEW Developer's Kit is part of the product line.



Let Gpf write the GUI you design

Using the powerful point and click visual programming environment of Gpf*, you can prototype, test and generate a complete OS/2 PM GUI in a few hours or days rather than the weeks or months required to hand code the same design. Even a relatively simple GUI can require writing thousands of lines of code, but with Gpf you simply draw your user interface on the screen. The integrated dialogue editor of Gpf permits actions and context sensitive help to be linked to controls as you create them. Gpf then generates error free ANSI C complete with embedded SQL statements.

Gpf is optimized to take full advantage of OS/2 PM, the most powerful and robust GUI system available. Since Gpf code directly accesses the PM API, there is no run time module to distribute with your application and no added overhead or royalties.

Gpf keeps the entire design definition in one file. This permits easy re-entry for updates as well as regeneration for different targets. 32 bit OS/2, 16 bit OS/2 and MS Windows code generation.

Gpf supports:

- Simple and direct linkage of the interface to program logic, built in or user defined functions.
- Direct association of help screens with controls, and complete integration into the PM Help Presentation Facility.
- Flexible use of Presentation objects (fonts, colors, etc.) with controls and windows (client area and frame).
- Simple inclusion of bitmaps for use on About screens, user-defined buttons, and menu or pulldown entries.
- Automatic embedded SQL statements to read OS/2 DataBase Manager tables, directly into combo or list boxes.
- Multi-thread programming.
- Multiple source file generation.
- Automatic creation of controls that scale with window size.
- Inclusion of user defined controls

Try us out

Order Gpf today for just \$995.⁹⁹

Call Gpf Systems Inc. at:

(203) 873-3300 or (800) 831-0017 - fax (203) 873-3302. Free demo software available

30 Falls Rd., P.O. Box 414, Moodus, CT 06469

* GUI Programming Facility

Gpf 2.0
Available Now
32 Bit Code Generation
CUA '91 Controls

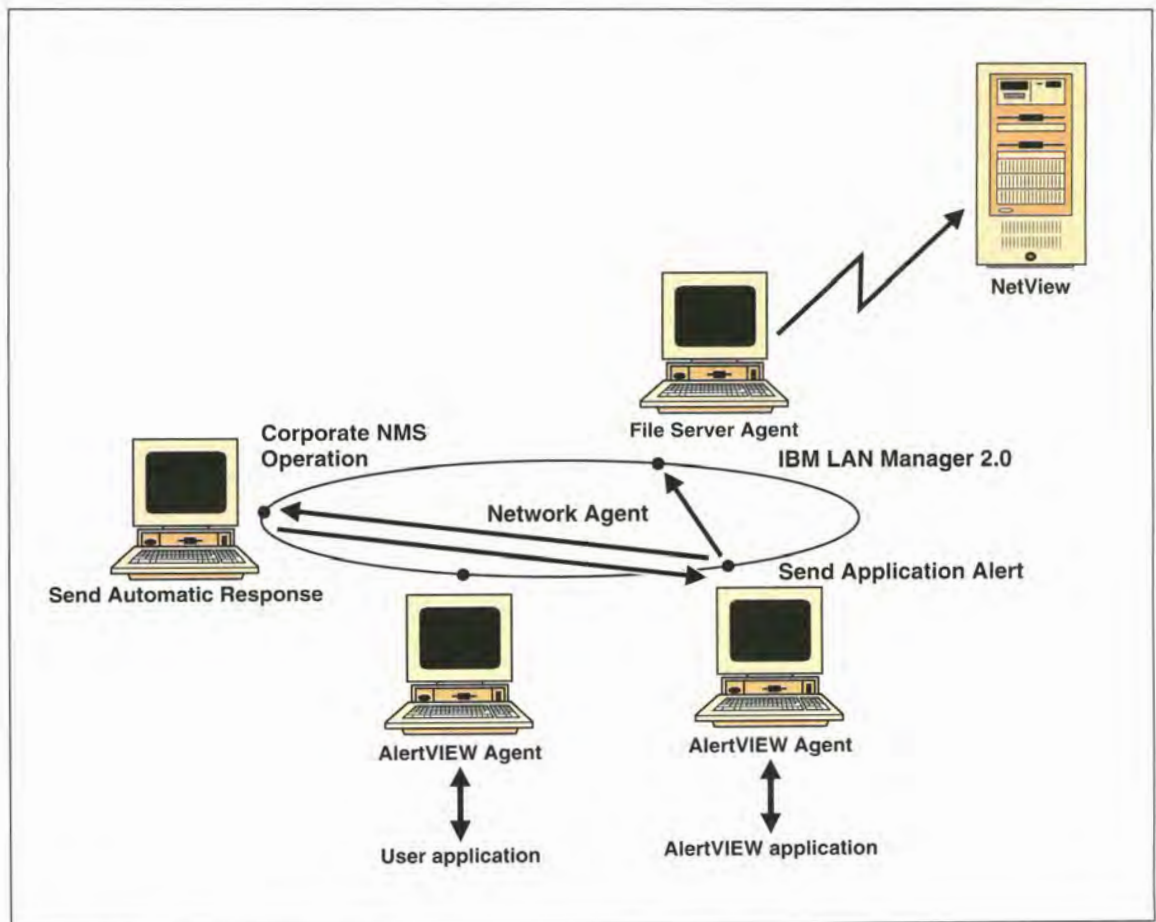


Figure 3: AlertVIEW Modules and IBM NetView

AlertVIEW Station (AVS)—This workstation agent can be loaded on any network station. It detects DOS, Windows, and OS/2 application errors, system calls, BIOS, and network operating system errors, NetBIOS or IPX communication errors, and so on.

AlertVIEW Manager (AVM)—This management console is used to remotely control the AlertVIEW agents, dynamically define alert filters, freeze and reboot workstations, and control application operations.

AlertVIEW Event Monitor (AVEM)—This is an enhanced event monitor designed to collect, display, and automatically respond to all AlertVIEW and other IBM SNA alerts.

AlertVIEW Anti-Virus (AVAV)—AVAV is an anti-virus agent used to detect, scan, remove, and alert the entry or focal point of a virus on any LAN station or server.

API DEVELOPER'S KIT SPECIFICATIONS

Hardware requirements—IBM PS/2 or PC or compatibles using PC-DOS 3.3 or higher, Windows 3.x, OS/2 1.21, 1.3, or 2.0.

LAN requirements—IBM Token Ring LAN supporting 802.2 interface or compatible or Novell Netware LAN running SPX/IPX. The alerts may be viewed on Shany's AlertVIEW Event Monitor or IBM's consoles.

Focal point—IBM LAN Manager 2.0 or LAN Network Manager 1.0, LAN Network Manager 1.1 or any other IBM focal point.

Memory requirements—The AlertVIEW API run-time application requires 5K of the workstation's RAM.

NetView support—Alerts may be forwarded to NetView by IBM LAN Manager 2.0 or LAN Network Manager 1.0 or 1.1.

Shlomo Touboul, Shany Inc., 2680 Bayshore Pkwy., Suite 104, Mountain View, Calif. 94043 (415) 694-7410, fax (415) 694-4728. Touboul is the president of Shany Inc. and Shany Ltd. in Natanya, Israel. After leading a communications software team in Fibronics, Touboul established Shany to develop a suite of network-enhanced products including AlertVIEW, Print+, Spool+, File+, Disk+, Queue+ and Remote Access. Touboul has a B.S.C in computer science from the Technion University, Haifa, Israel.

REFERENCES

IBM LAN Manager (Version 2.0) User's Guide (IBM. Doc. SX27-3710-04)

Token Ring Network Problem Determination Guide (IBM. Doc. G520-6575-00)

Local Area Networks (LAN) IBM AlertVIEW Manager Administrator's Guide, Shany Inc.

AlertVIEW Manager Administrator's Reference, Shany Inc.

AlertVIEW API Reference, Shany Inc.



OPEN SHUTTER **Screen Capture & Conversion For OS/2**

MORE FUNCTION, LESS PRICE....WHAT A CONCEPT!

At One Up Corporation, abundance in function without abundance in price is paramount. Consequently, you'll find our screen capture application, Open Shutter, provides the flexibility you need to capture, modify, and output your desktop images at a price you'll want to photograph!

CAPTURE YOUR IMAGES

Our easy-to-use capture process allows you to select any rectangular area, window, or the entire desktop and capture with a single user-defined keystroke or mouse click.

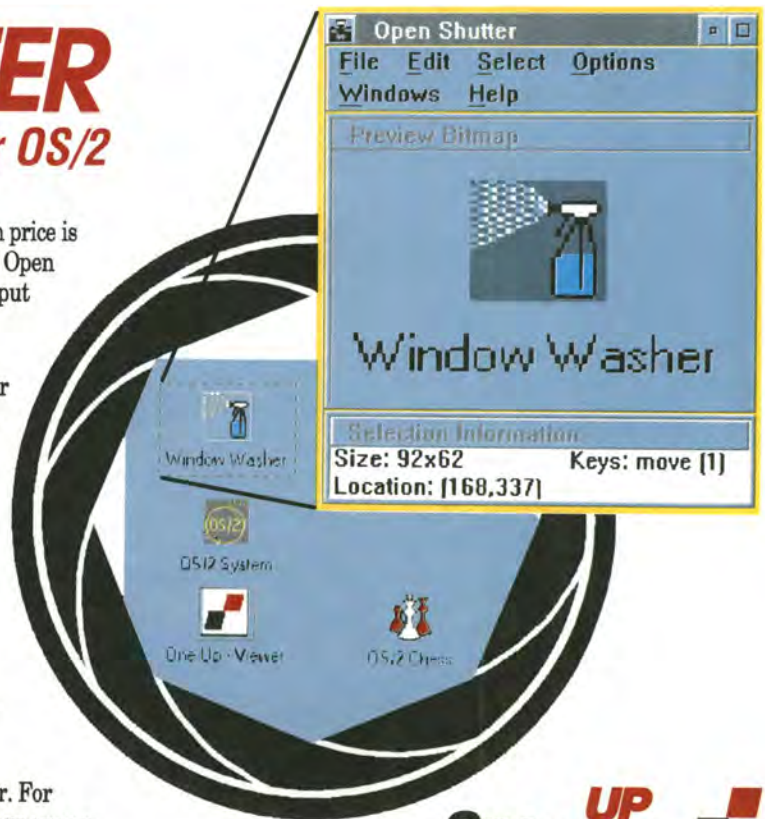
MODIFICATIONS MADE SIMPLE

Rotate your image at any angle, map your colors and modify your color palette, and stretch or compress your image to any dimension you need. Preview/compare multiple modified versions of your captured image prior to output.

WHAT ABOUT OUTPUT?

Open Shutter offers a varied selection of output devices including printer and clipboard. For soft copy, save your image as an OS/2 or Windows BMP, ICO, or metafile. Also, save as an OS/2 pointer, a Windows cursor, or PCX, TIFF, GIF, PICT formats and more! So don't get discouraged by high price applications that don't deliver. For only **\$59.95**, Open Shutter offers you a cost conscious solution for your screen capture requirements. Ask us about our competitive upgrade price, too.

OS/2 is a registered trademark of IBM Corporation
Windows is a registered trademark of Microsoft Corporation



One UP Corporation
 1603 LBJ FREEWAY, SUITE 860
 DALLAS, TEXAS 75234
 1-800-678-01UP

Tools

PMfax: Device-Independent Fax Services for OS/2 2.0



Mark Ahlstrom

by Mark Ahlstrom and Bruce Keller

The fax services described in this article make it easy to send fax documents from all applications that run on OS/2 2.0. The complexities of creating fax documents and dealing with fax hardware and protocols are encapsulated in PMfax™ (also distributed as FaxWorks™ OS/2), which allows applications to work with most PC fax hardware in stand-alone and LAN configurations.

Fax machines are now installed in most business offices. Sending a fax document is almost as easy as dialing a telephone, and fax transmission provides immediate document delivery for the price of a phone call. For these reasons, fax transmission is increasingly becoming the method of choice for document delivery.

When faxing a paper document between traditional fax machines, the sending fax machine scans the original document and transmits the scanned image over the telephone line using a standard fax transmission protocol, somewhat like the data transfer protocols used to transfer data files between computers. The receiving fax machine receives and prints the scanned image.

A computer with a fax modem or fax coprocessor board can send and receive fax documents. Rather than scanning a paper document, fax software converts a document to a form suitable for fax transmission, without the blurring often produced by scanning. When used to receive, fax software can display a document on the computer screen or print it on a computer printer, using plain paper instead of the heat-sensitive paper often used in fax machines. Computer fax hardware is readily available, and many modems now include built-in fax capabilities.

For a software developer, the implications of fax technology become obvious if you think of a fax machine as a printer connected by a telephone line. In other words, an application can print to millions of fax machines and computer fax devices throughout the world. The difficulty is that most applications don't know how to deal with the intricacies of creating and sending fax documents.

This article demonstrates the use of fax services that make it easy to create and send fax documents from applications. Just as OS/2 printer drivers provide a device-independent way for applications to use printers, PMfax provides a device-independent way for applications to send fax documents with different types of fax modems and fax boards.

PMFAX AND FAX SOFTWARE

PMfax is an OS/2 fax software product consisting of an application program and a printer driver.

The PMfax application, shown in Figure 1, is used to manage fax documents. With its viewing and editing features, the user can create documents using the Clipboard, text, bitmap, or PCX files, screen captures, other fax documents, keyboard input, and mouse-based drawing and editing tools. PMfax sends, receives, and prints documents using background threads. Supported fax hardware includes Class 1 and Class 2 fax modems, SatisFAXtion™ and Brooktrout™ boards. Using OS/2 time-critical threads, an optimized device driver for fax modems, and other OS/2-specific features, PMfax provides background fax operation. PMfax is available in stand-alone and LAN versions. The fax services described in this article can run on any OS/2



Bruce Keller

machine, including LAN workstations and servers.

Once the PMfax printer driver is installed, applications can create fax documents with their File/Print command. Documents can also be printed from WIN-OS/2 and DOS applications. The printer driver passes the fax documents to PMfax. When the printer driver has created a fax document, a pop-up window allows users to send it as is or save it for further editing. Multiple documents can be spooled for transmission.

While most OS/2 services are provided through APIs, there are several advantages to using an OS/2 printer driver to create and send fax documents. With the OS/2 2.0 Workplace Shell, the user can print by dragging a file object to a printer object. Because OS/2 printer drivers are a shared system resource, the printer device can be shared over a LAN and accessed by both OS/2 and non-OS/2 workstations. In addition to making direct calls to the printer driver, applications can print to a printer driver by copying to a logical printer device such as LPT2. DOS and Windows applications can also print to the OS/2 printer driver this way.

GENERAL FAX SERVICES ON OS/2 2.0

When adding a fax interface to applications, it is best to support fax hardware from many vendors instead of a single product. It is, however, difficult to support even one type of hardware. Building fax capabilities into an application can be complex for several reasons. Many computer fax products are available, and there are several incompatible fax command language standards in common use. Also, fax data formats and protocols are relatively complex, and each page in a fax document must be created and compressed according to specific standards. Finally, on a multitasking operating system such as OS/2, fax services must be viewed as a shared resource that can be simultaneously used by multiple applications, even while sending and receiving occurs in the background.

For example, fax modems that use the Class 1 fax command set are supposed to adhere to an international standard, but there are many

subtle differences between Class 1 fax modems. Although other fax modems are based on the Class 2 fax command set, this set was never approved as a standard; it is now being replaced by the incompatible Class 2.0 standard. Coprocessor boards such as Intel's SatisFAXtion are even more challenging to use on OS/2 because they use proprietary fax I/O interfaces instead of COM ports. The situation is further complicated by variations in the many fax machines with which users must communicate. PMfax avoids much of this complexity with its hardware-independent interface.

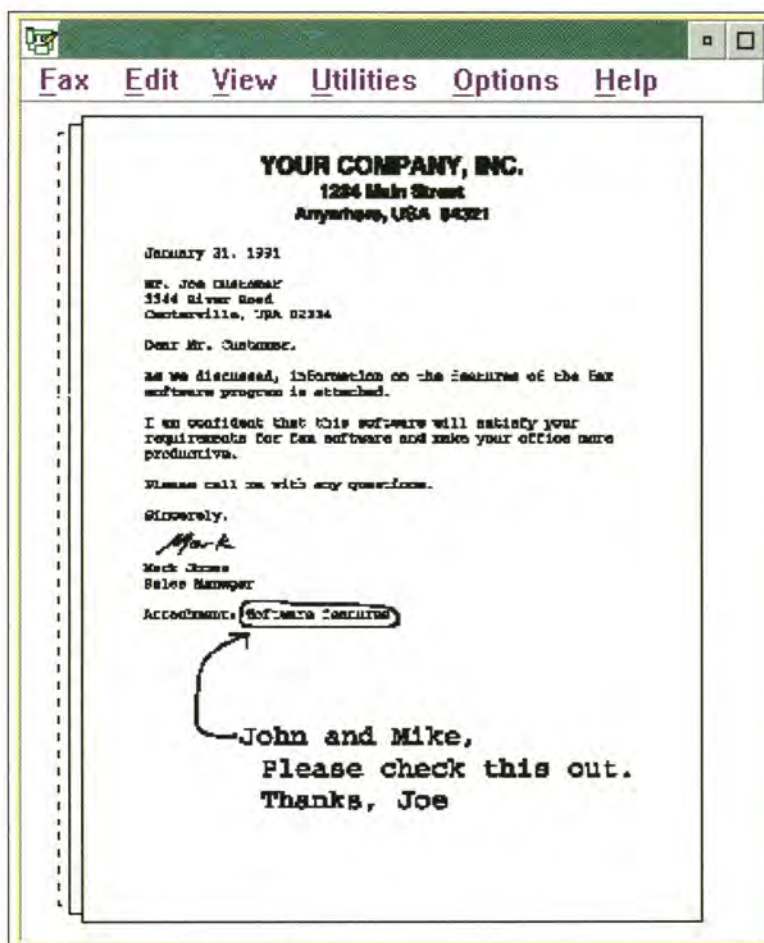


Figure 1: PMfax application program

FAX SPOOLING FOR TEXT OUTPUT

When the PMfax printer driver is installed, its printer object is associated with a logical printer device such as LPT2. Thereafter, ASCII text copied to this logical device is automatically converted into a fax document.



OS/2 Adobe fonts are used to convert text characters into high-quality printed representations of the characters on the fax pages.

>>TO= name, company, faxnumber	Specifies recipient; multiple >>TO lines allowed
>>FROM= name, company, phonenumber, faxnumber	Changes sender's cover sheet information
>>INFO= comment, heading, notes, bitmap	Changes header line and cover sheet items
>>AT= time, date	Specifies date and time for delayed fax transmission
>>FONT= font, points, tabchars	Selects any OS/2 Adobe font for text conversion
>>PAGE= length, orientation	Changes page length and landscape/portrait orientation
>>FILE= filename	Stores fax as a TIFF-F file without making a fax log entry

Table 1: Printer driver commands in PMfax 1.10

```
>>FROM=Joe Sender, Joe's Company Inc., 555-7788, 555-7766
>>INFO=Comment text for cover page, Joe's Co. Inc., Test,*
>>AT=5p, 3may92
>>TO=Bill Smith, Smith Corp.,555-5566
>>TO="Jim "JB" Brown", Brown & Sons, 555-9999
>>TO=Mark Jones, "Universal Fax, Inc.", 1 (213) 555-1122
```

This text will appear in the default font (Courier 12 point).

```
>>FONT=,16
```

Or you can make the current font bigger (this is Courier 16 point).

```
>>FONT=Courier Bold Italic,18
```

This text will be bigger, bolder and italic. You can have as many lines of text as you want. When the end of a page is reached, a new page is started. The default page length is a job property of the printer object, or can be changed on a page-by-page basis with the >>PAGE command.

```
>>FONT=Helvetica
```

This will be in Helvetica font, still in the 18 point size.

```
>>FONT=Times New Roman, 90
```

This is HUGE!

```
>>FONT=*,*
```

And back to the default font (Courier,12 point).

A user can create a fax document by using the COPY command to write a text file to the logical printer device. The printer driver will display a "Send Fax" window that specifies destination and cover sheet information and allows the user to send the document. Any OS/2, Windows, or DOS program or command file can also create a fax by copying text to the logical printer device.

With PMfax, special printer driver command lines can be included in the text to automatically send the fax document to one or more recipients, as shown in Table 1. If any >>TO command lines are included in the text, the "Send Fax" window is suppressed and no additional user action is required to transmit the document. Additional command lines can be used to specify special values for a document's font, page layout, cover page information and fax header line when they vary from the default, as shown in Figure 2.

When the file in Figure 2 is copied to the device on which the printer driver is installed, a fax document is created and spooled for transmission to three recipients. An asterisk (*) parameter causes the default value to be used; font parameters can be omitted to leave a value unchanged. The user can set default cover sheet and header line strings in PMfax, and can adjust other transmission parameters such as how many times to retry if the recipient's fax machine is busy.

FAX SPOOLING FOR FONTS AND GRAPHICS

Developers can directly spool fax documents by adding a few additional calls to the normal printing code. As demonstrated in Figure 3, even applications that use GPI calls for fonts and graphics can use PMfax's printer driver commands.

Fax creation and spooling can be integrated parts of an application: in addition to the File/Print command, the application can add a File/Fax command that creates and spools the fax document. Applications with print merge features that generate customized form letters can add a fax merge feature that will automatically fax the customized letter to each recipient. Applications do this by using >>TO= strings and other printer driver commands

Figure 2: Specifying special values for document elements



```

VOID
spool_fax(
    VOID )
{
    HDC hdc;
    HPS hps;
    DEVOPENSTRUC dop;
    SIZEL sizl;
    PCHAR title = "MyFaxProg";
    PCHAR from = ">>FROM=Sam Sender, Sam's Company, 555-1111, 555-2222\r\n";
    PCHAR to = ">>TO=Rick Receiver, Rick Inc., 555-3333\r\n";
    PCHAR info = ">>INFO=Please read this fax, Sam Co., Cover.bmp\r\n";
    POINTL ptl;
    LONG n;
    ULONG faxid;

    /* Fill in device open structure with selected printer information */
    memset( &dop, 0, sizeof( DEVOPENSTRUC ) );
    dop.pszLogAddress = printer_logname;
    dop.pszDriverName = printer_drvname;
    dop.pdriv = printer_drvdata;
    dop.pszDataType = "PM_Q_RAW";
    dop.pszComment = title;

    /* Open printer device context */
    hdc = DevOpenDC( hab, OD_QUEUED, "*",
        sizeof( dop ) / sizeof( PVOID ), (PDEVOPENDATA)&dop, NULL );

    /* Open printer presentation space (if needed for Gpi calls) */
    sizl.cx = sizl.cy = 0;
    hps = GpiCreatePS( hab, hdc, &sizl,
        GPIF_DEFAULT | GPIT_NORMAL | GPIA_ASSOC | PU_LOENGLISH );

    /* Signal start of document */
    DevEscape( hdc, DEVESC_STARTDOC, (LONG)strlen( title ), title, NULL, NULL );
    DevEscape( hdc, DEVESC_RAWDATA, (LONG)strlen( from ), from, NULL, NULL );
    DevEscape( hdc, DEVESC_RAWDATA, (LONG)strlen( to ), to, NULL, NULL );
    DevEscape( hdc, DEVESC_RAWDATA, (LONG)strlen( info ), info, NULL, NULL );

    /* Generate the document (with Gpi or more DevEscape raw text calls) */
    ptl.x = 100;
    ptl.y = 100;
    GpiMove( hps, &ptl );
    ptl.x = 700;
    ptl.y = 1000;
    GpiBox( hps, DPO_OUTLINE, &ptl, 50, 50 );
    ptl.x = 200;
    ptl.y = 900;
    GpiCharStringAt( hps, &ptl, strlen( title ), title );

    /* Signal end of document and get fax log id number */
    n = sizeof( faxid );
    DevEscape( hdc, DEVESC_ENDDOC, 0L, NULL, &n, (PBYTE)&faxid );

    /* Close printer */
    GpiDestroyPS( hps );
    DevCloseDC( hdc );
}

```

Figure 3: OS/2 printing to the PMfax printer driver with automatic spooling for transmission

before issuing the application's standard printing commands, as shown in Figure 3.

PLANNED EXTENSIONS

Several additional features are planned for PMfax. Most developers prefer to approach fax services like printer services, allowing the fax application to manage fax delivery in the same way print spoolers manage printing. But in some cases developers need additional control. Functions will be added that allow applications to check the status of their fax jobs, schedule additional delivery attempts, and delete documents after transmission.

The Keller Group plans to add printer driver commands for placing graphics files on the fax page and appending other fax document files. These commands will allow any application, including DOS applications and command files, to create complex fax documents. In addition, enhanced LAN versions of PMfax will soon be available, with versions that support simultaneous use of multiple fax lines in each fax server.

SUMMARY

Faxing has become common for document delivery, and many applications can benefit from transmitting data by fax. Through fax services, applications can operate transparently with most PC fax hardware products in stand-alone, LAN, and multiline configurations.



Bruce Keller, Keller Group Inc., 8600 Jewel Avenue North, Stillwater, Minn. 55082, (612) 429-7273, fax (612) 653-1987. Keller is a founder and vice president of Keller Group Inc. He previously founded Airplan Systems (acquired by Continental Airlines), where he pioneered the use of OS/2 LANs for high-volume automatic fare-checking services. Previously, he worked for Cray Research, where he ported the UNIX kernel to the Cray-2 computer. He holds a B.A. in mathematics from St. Olaf College.

Mark Ahlstrom, Keller Group Inc., 8600 Jewel Avenue North, Stillwater, Minn. 55082, (612) 429-7273, fax (612) 653-1987. Ahlstrom is a founder and vice president of Keller Group Inc., which develops device drivers and APIs distributed by several fax manufacturers. Previously, Ahlstrom was a founder of Airplan Systems and a member of the artificial intelligence department at the Honeywell Computer Sciences Center. He holds a B.S. in biochemistry and an M.S. in biomedical engineering, both from the University of Wisconsin at Madison.

REFERENCES

Fannin, Michael. "Open+Build™: A Voice, Fax, and Telecommunications Front-End For Databases," *IBM Personal Systems Developer* (Spring 1992): 37-45.

Reich, David. "Programming Printing Under OS/2," *IBM Personal Systems Developer* (Winter 1992): 85-95. This article describes the OS/2 printer subsystem architecture.

FaxWorks OS/2 User's Manual (FaxWorks is a version of PMfax published by SofNet Inc.) The manual can be obtained by calling (404) 984-8088.

PMfax Developer Support Program & Developer Licensing, available from Keller Group, lists services and pricing for developers who wish to distribute the PMfax engine as a component of their products.

IBM OS/2 Programming Guide Vol. III: Graphics Programming Interface (IBM Doc. 10G6495) provides information on how to prepare graphical output for printing and print job submission and manipulation.

Tools

Why PL/I?

by Davinder S. Athwal

Readers may be wondering what an article on PL/I is doing in a magazine devoted to OS/2. ISVs most likely use C and are reasonably happy with it, while corporate in-house programmers may or may not have used PL/I in a mainframe environment. Either way, the idea of using PL/I on the desktop may seem strange at first. But before you skip ahead, look more closely at PL/I's power and productivity, and its new 32-bit implementation for OS/2, PL/I Package/2. PL/I Package/2, introduced in September 1992, is the newest language product of the SAA AD/Cycle family. A personal version, PL/I Workstation/2, became available in December 1992.

Developers familiar with PL/I may wish to skip to the heading "PL/I Package/2 and PL/I Workstation/2 Products" on page 88. The following is an overview of PL/I's functions and strengths.

PL/I OVERVIEW

PL/I is a free-form high-productivity language with a variety of features that support scientific, engineering, commercial, and system programming tasks. It is designed to offer robustness, machine independence, structured programming constructs, powerful exception handling, dynamic storage, a wide variety of other data types and aggregates (arrays, structures, and unions), I/O capabilities, and many built-in functions.

Free-Form Structure

PL/I is completely free-form and has no reserved keywords; it determines the meaning of keywords by context. The programmer can

concentrate on program logic rather than worrying about word use. For example, in PL/I it is valid to declare a variable named **AREA**, despite the fact that **AREA** is used as a PL/I keyword. Because PL/I offers many defaults, developers need not be aware of all the language's features to use it confidently.

Machine Independence

PL/I data types are hardware independent. For example, an integer is defined as having a base and a precision. Therefore, **FIXED BINARY(12)** declares a binary integer of 12 data bits and **FIXED DECIMAL(31)** declares a decimal integer of 31 decimal digits even if the machine does not support packed decimal data. All PL/I implementations support all language data types, even those the hardware cannot support directly; this makes for truly portable code.

Program Structure

PL/I is a block-oriented language consisting of packages, procedures, begin blocks, and statements. Because PL/I allows you to control the scope, visibility, shareability, and lifetimes of variables, exceptional conditions, and procedures, you can produce highly modular, independently testable, and reusable code.

Structured programming constructs

Besides providing the basic constructs of most other languages, PL/I adds significant power, which allows program logic to be highly modular and well structured for greater reliability and easier development, maintenance, and extensibility. PL/I has extremely powerful case statements and



Davinder Athwal



looping constructs that permit combinations of until, while, and repetition specifications. It also supports multiple specifications of these loops.

Data Types

A variety of data types can be represented and manipulated in PL/I, making the language appropriate for a range of applications. Arithmetic data may be real, complex, floating-point, fixed-point, signed, unsigned, binary, decimal, numeric, or editable picture data. String data may be varying, nonvarying, character, bit, graphic, run-time adjustable, or compile-time fixed length strings. Program control data also supports pointer and offset locators, adjustable or fixed size areas for grouping dynamic allocations, entry data (function pointers), and file, label, and format data. AREA data may be written out to the disk and read back at a later date, even if the allocated data within an AREA is a linked list.

```
Declare 1 Year(1901:2100),
      3 Month(12),
      5 Temperature,
        7 High decimal fixed(4,1),
        7 Low decimal fixed(4,1),
      5 Wind_velocity,
        7 High decimal fixed(3),
        7 Low decimal fixed(3),
      5 Precipitation,
        7 Total decimal fixed(3,1),
        7 Average decimal fixed(3,1),
      3 * char(0);
```

Figure 1: Array of structures to hold weather data

The variety of data types provided in PL/I allows developers to concentrate on the application logic instead of finding ways to circumvent deficiencies in the programming language.

Data Arrays, Structures, and Unions

In PL/I, data items can be either single data elements or data aggregates and can be referred to either collectively or individually. Data aggregates can be arrays, structures, unions, arrays of structures or unions,

structures or unions of arrays, and so on. This flexibility provides a strong foundation for the description and manipulation of data. Figure 1 shows how an array of structures used to hold meteorological data for each month of the 20th and the 21st centuries might be declared. In Figure 1, the weather data for July 1991 is contained in the element `Year(1991,7)` of the array of structures `Year`. Portions of this data can be referred to by `Temperature(1991,7)` or `Year.Month.Precipitation(1991,7)`. The upper and lower bounds of an array can be negative, zero, or positive. PL/I does not force a lower bound of zero or one, which can contribute to the readability, reliability, and maintainability of a program.

Data Manipulation

Data may be easily manipulated as individual items or as an aggregate. A single invocation of the `SQRT` function, for example, can take the square root of an entire array, producing an identical array containing square root values in the corresponding elements. In other languages, in contrast, array and aggregate operations must be broken into elementary operations, which results in less understandable and less maintainable code.

Built-in Functions

PL/I provides almost 200 built-in functions, pseudovariables, and subroutines that manipulate scalar and aggregate data. These functions cover character, bit, and graphic string data, arithmetic computation, mathematical computation, floating point inquiries, floating point manipulation, integer manipulation, precision handling, array manipulation, storage control, condition handling, input and output, and date and time manipulation.

Restricted Expressions

Restricted expressions are expressions, of any complexity, evaluated at compile time. Figure 2 shows a static "hex to character" translate table. The compiler evaluates complex expressions involving these nested built-in functions during compilation. The code, entirely portable across platforms, is character set (ASCII vs. EBCDIC) and code page independent. This operation is much more complicated in other languages.

Storage Classes, Control, and Dynamic Storage Allocation

PL/I offers four storage classes: AUTOMATIC, STATIC, CONTROLLED, and BASED.

Application objects' data type, representation, nature of use, and so on normally dictate the type of storage class used for each. With PL/I, programmers can dynamically allocate and free BASED and CONTROLLED storage. All data, not only static data, may be automatically initialized for all storage classes upon allocation. This leads to more readable and reliable code, as the initial values are part of the data declaration and are automatically assigned.

Condition Handling

PL/I can handle a variety of program execution problems. Problems can be detected by either the hardware (e.g., ZERODIVIDE), PL/I (e.g., CONVERSION), the operating system, or other software. These facilities allow you to detect exceptional conditions and take appropriate corrective action, as well as produce applications that provide non-stop operations. You can even handle user initiated attention interrupts and normal and abnormal program termination.

Program Checkout

There are several built-in program checkout conditions that automatically check a program for out of range array subscripts, strings, and substrings, as well as data that exceeds declared or machine supported precisions. These features help produce defect-free applications.

Input and Output

PL/I programs can process various sizes and types of records, access many different devices, edit and validate several types of input and output data, and produce report files. There are two types of input/output (I/O) in PL/I, record-oriented I/O and stream I/O.

Record-oriented I/O logically transmits one record at a time without performing any data conversion. These files may be stored in indexed, relative, or consecutive formats.

Stream I/O deals with a logically continuous stream of characters (that may exist externally as physical records), transmitting one data item

at a time while performing conversions between the internal and external forms. Data items may be transmitted as a list, edited list, annotated list, or a combination thereof.

While PL/I can handle mundane operations such as file opening and closing automatically, it offers programmers precise control over I/O operations, status, and exceptional conditions. I/O condition handling features include end of file reached, end of page reached, record not found, record truncated, I/O errors, file cannot be found, file open, key for which record was



```

dcl
  1 *   union static,

      2 cased_hex(0:255) fixed bin(8) unsigned,

      2 * bit( 8*256 )
  init( (
      ( copy('ff'b4,256) )
      & (   copy('ff'b4,index(collate(),'0')-1)
          || '00010203040506070809'b4
          || copy('ff'b4,256-(10+index(collate(),'0')-1)) )
      & (   copy('ff'b4,index(collate(),'a')-1)
          || '0a0b0c0d0e0f'b4
          || copy('ff'b4,256-(6+index(collate(),'a')-1)) )
      & (   copy('ff'b4,index(collate(),'A')-1)
          || '0a0b0c0d0e0f'b4
          || copy('ff'b4,256-(6+index(collate(),'A')-1)) )
      ) );

```

Figure 2: Use of restricted expressions in initialization of static data

not found, source that caused conversion error, character within source that caused conversion error, file associated with the I/O condition, number of current page, number of current line within a page, and bad data. Built-in functions help retrieve and change condition related data.

Macro Facility

PL/I's macro facility permits conditional compilation and modification of PL/I source code before compilation. For example, a programmer might include certain code during development and testing, and exclude it from a production program without reediting the original. Different parts of source code can also be platform-specific.



PL/I PACKAGE/2 AND PL/I WORKSTATION/2 PRODUCTS

PL/I Package/2 and PL/I Workstation/2, 32-bit products for OS/2 2.0, have several new features that help developers write Presentation Manager (PM) applications using the SAA PL/I common programming interface and extensions that work with OS/2 2.0. These products also allow users to write non-Presentation Manager applications that can use built-in PM features to provide a graphical interface. The hardware and software required to run the products includes an IBM PS/2 or compatible machine with OS/2 2.0 or later.

PL/I Workstation/2 provides all the features of PL/I Package/2, except VSAM-like record-level I/O support and PL/I language levels of SAA and OS. Language level SAA2, of which SAA is a subset, is the only language level supported by PL/I Workstation/2.

PL/I Package/2 and Workstation/2 also include facilities to enhance programmer productivity, program reliability, installation standardization, and control. They address numerous user requirements submitted by PL/I customers and SHARE and GUIDE groups in the U.S. and around the world, and implement most features from the American National Standard PL/I General Purpose Subset (ANSI X3.74-1987).

The PL/I compiler produces high performance optimized object code that is reentrant and can be recursive. It meets the requirements of applications that exploit OS/2 and the Presentation Manager, and can help develop and maintain host applications.

Programming Support

Both PL/I products, consisting of the PL/I compiler and run-time libraries, fully support IBM WorkFrame/2. Several language extensions allow PL/I programs to communicate with programs written in other languages, including C Set/2.

The PL/I products support the OPTLINK and the SYSTEM inter-procedure linkages, also used in C Set/2. In Figure 3, a PL/I program plays a tune using OS/2 services with no

special coding required. It also shows use of named constants, such as `whole`, and restricted expressions in the initialization of the `notes` two-dimensional array. If one were to change the named constant `whole`, the compiler would automatically recalculate all dependent values and declarations.

Figure 4 shows a PL/I program that converts hex strings to character. The object code produced by the compiler is as good as if it were hand-coded.

Figure 5 shows a PL/I Presentation Manager program. No special coding is required; all communication with PM is direct.

Hardware Support

PL/I Package/2 and Workstation/2 exploit 386™ and 486™ processors including floating point hardware. Scaled and unscaled binary and decimal arithmetic are also fully supported.

Programmer Productivity Features

The following features significantly accelerate program development and maintenance and increase program reliability:

- Features that detect or, in some cases, prevent the introduction of defects. Under a compiler option, for example, undeclared variables can be flagged as errors.
- Extensive use of ON-units, which allow sophisticated exception handling and recovery. A new condition, `INVALIDOP`, allows you to intercept and correct various IEEE floating point conditions. Another condition, `STORAGE`, lets you intercept out-of-storage situations.
- Clear and precise compile-time and run-time diagnostics, which pinpoint the original file and the line causing the problem. As one of the several improvements over S/370 OS PL/I, compiler listings show modified (as well as referenced and unreferenced) variables.

Usability Features

Several PL/I features allow you to tailor the products to your specific needs:



```
%process;
chimes: proc options(main reorder);

  dcl ( rest value( 0 ),
        g4 value( 392 ),
        c5 value( 523 ),
        d5 value( 587 ),
        e5 value( 657 ),
        whole value( 800 ) ) fixed bin(31);
  dcl notes(19,2) static nonassignable fixed bin(31)
  init( e5, (whole/2),
        c5, (whole/2),
        d5, (whole/2),
        g4, (whole),
        rest, (whole/2),
        g4, (whole/2),
        d5, (whole/2),
        e5, (whole/2),
        c5, (whole),
        rest, (whole/2),
        e5, (whole/2),
        c5, (whole/2),
        d5, (whole/2),
        g4, (whole),
        rest, (whole/2),
        g4, (whole/2),
        d5, (whole/2),
        e5, (whole/2),
        c5, (whole) );
  dcl i fixed bin(31);

  dcl playnote entry( fixed bin(31), fixed bin(31) )
    returns( fixed bin(31) optional )
    external( 'DOSBEEP' )
    options( byvalue LINKAGE(SYSTEM) );

  dcl restnote entry( fixed bin(31) )
    returns( fixed bin(31) optional )
    ext( 'DOSSLEEP' )
    options( byvalue LINKAGE(SYSTEM) );

  do i = lbound(notes,1) to hbound(notes,1);
    if notes(i,1) ^= 0
      then call playnote( notes(i,1), notes(i,2) );
      else call restnote( notes(i,2) );

  end;
end;
```

Figure 3: Program playing a tune



```

*process langlvl(saa2) or('|') not('^^')
*process dft( byvalue reorder ) opt(2)
*process prefix ( noFixedOverflow ) limits(extname(31));

hex2char: procedure(addr_out, length_in, addr_in);
  declare addr_out      byvalue pointer;
  declare length_in     byvalue fixed bin(15);
  declare addr_in       byvalue pointer;

  declare
    1 * based,
    2 first_half      unsigned fixed bin(8),
    2 next_half       unsigned fixed bin(8);

  declare based_bin8    unsigned fixed bin(8) based;
  declare left_half     unsigned fixed bin(8);
  declare right_half    unsigned fixed bin(8);
  declare bytes         fixed binary(31) initial (0);

  /* note that this variable is static */
  /* it is also code page independent */

  dcl
    1 *  union static nonassignable,
    2 cased_hex(0:255) fixed bin(8) unsigned,
    2 * bit( 8*256 )
    init( (
      ( copy('ff'b4,256) )
      & (   copy('ff'b4,index(collate()),'0')-1)
          || '00010203040506070809'b4
          || copy('ff'b4,256-(10+index(collate()),'0')-1)) )
      & (   copy('ff'b4,index(collate()),'a')-1)
          || '0a0b0c0d0e0f'b4
          || copy('ff'b4,256-(6+index(collate()),'a')-1)) )
      & (   copy('ff'b4,index(collate()),'A')-1)
          || '0a0b0c0d0e0f'b4
          || copy('ff'b4,256-(6+index(collate()),'A')-1)) )
    ) );

  do while( bytes < length_in );

    left_half
      = raise2( cased_hex( addr_in->first_half ), 4 );
    right_half
      = cased_hex( addr_in->next_half );

    addr_out->based_bin8
      = ior( left_half, right_half );

    addr_out = addr_out + 1;

    addr_in = addr_in + 2;
    bytes = bytes + 2;
  end;
end;

```

Figure 4: Hexadecimal to character conversion routine



```

*PROCESS LONGLVL( SAA2 ) INCLUDE(EXT (INC)) MACRO;
*PROCESS LIMITS(EXTNAME(31)) A( F ) S F( W, 20 );
*PROCESS MARGINS(2,100);

PM_App_Package: Package exports (PM_App);

/* INCLUDE files */

%NOPRINT;
%DCL INCL_WIN CHAR;
%INCL_WIN = 'Y';
%INCLUDE OS2;
%PRINT;

PM_App: Proc options( main );

/* Constants */

dcl title char( 30 ) static init ( 'PL/I Package/2 Demonstration'Z );
dcl flags fixed bin(31) static init(( FCF_TITLEBAR + FCF_SIZEBORDER +
    FCF_SYSMENU + FCF_MINMAX + FCF_SHELLPOSITION + FCF_TASKLIST +
    FCF_VERTSCROLL + FCF_HORZSCROLL ));

/* Local variables */

dcl hwndClient          HWND;
dcl AnchorBlockHandle   HAB;
dcl MessageQueueHandle  HMQ;
dcl MessageQueue        QMSG;
dcl WindowHandle        HWND;

/* Code */
AnchorBlockHandle = WinInitialize( 0 );
MessageQueueHandle = WinCreateMsgQueue( AnchorBlockHandle, 0 );

Call WinRegisterClass
    ( AnchorBlockHandle,
      addr( title ),
      PM_App_WinProc,
      CS_REDRAW,
      0 );

WindowHandle = WinCreateStdWindow(
    HWND_DESKTOP,
    WS_VISIBLE,
    addr( flags ),
    addr( title ),
    addr( title ),
    0,
    null(),
    0,
    addr( hwndClient ));

```

Figure 5: A Presentation Manager program (continued on page 92)



```

do while ( WinGetMsg( AnchorBlockHandle, addr( MessageQueue ),
                    null(), 0,0 ) ^= 0 );
    Call WinDispatchMsg( AnchorBlockHandle,addr( MessageQueue ));
end;

Call WinDestroyWindow( WindowHandle );
Call WinDestroyMsgQueue( MessageQueueHandle );
Call WinTerminate( AnchorBlockHandle );
END;

PM_App_WinProc: proc( WinHandle, msg, mp1, mp2)
    options(byvalue linkage(system)) returns (MRESULT);

/* Parameters */

Dcl WinHandle      HWND;
Dcl msg            ULONG;
Dcl mp1            MPARAM;
Dcl mp2            MPARAM;

/* Local Variables */

dcl rectangle      RCL;
dcl hps            HPS;
dcl hello char(12) static init ( 'Hello World' );

/* Code */

select (msg);
when( WM_PAINT )
do;
    hps = WinBeginPaint(WinHandle, null(), null());
    Call WinQueryWindowRect(WinHandle, addr(rectangle));
    Call WinDrawText(hps,12,addr(hello),addr(rectangle),0,0,DT);
    Call WinEndPaint(hps);
end;

when( WM_ERASEBACKGROUND )
return ( TRUE_PTR );

when( WM_BUTTON1DOWN )
do;
    Call WinSetActiveWindow( HWND_DESKTOP, WinHandle );
    Call WinAlarm( HWND_DESKTOP, WA_NOTE );
    display( 'Button 1 is pressed' );
    return( TRUE_PTR );
    return{ WinDefWindowProc( WinHandle, msg, mp1 } };
end;

```

Figure 5: A Presentation Manager program (continued on page 93)



```

when( WM_BUTTON2DOWN )
do;
  Call WinSetActiveWindow( HWND_DESKTOP, WinHandle );

  /* sound a different tone */
  Call WinAlarm( HWND_DESKTOP, WA_ERROR );

  display( 'Button 2 is pressed' );
  return( TRUE_PTR );
  return{ WinDefWindowProc( WinHandle, msg, mp1 } };
end;

otherwise
  return( WinDefWindowProc( WinHandle,msg,mp1,mp2 ));
end;
return( null() );
end;
end PM_App_Package;

```

Figure 5: A Presentation Manager program (continued from page 92)

- Compiler options that can be specified in environment variables, on the command line, or in the *PROCESS statement.
- File extensions, location and search tailorability for INCLUDE and macro processing
- Ability to create dynamic link libraries.

National Language Support (NLS)

Both PL/I Package/2 and Workstation/2 are NLS enabled and provide national currency symbol support in picture data. Identifier names can use European alphabetic characters such as "ü" or "ñ."

Compile-time facilities

There are many compile-time facilities in PL/I, including:

- Control of the degree to which the compiler optimizes source code
- Diagnostic message control, which permits specification of message limit and softening, strengthening, or suppression of messages from the compiler
- Control of language level enforcement, application of language related defaults,

dynamic condition prefix enablement and disablement, language rules enforcement, compiler listings, and alternate code points for OR and NOT symbols.

Input and Output Facilities

Edit-, list-, and data-directed stream I/O is supported for byte-stream and record files. In addition to record-level I/O support for byte-stream consecutive data sets, PL/I provides direct and sequential read, write, and update support for VSAM-like indexed, relative, and consecutive data sets.

Printer destined files are also supported; they may be printed locally or remotely on a host system.

The display statement is supported in Presentation Manager and non-Presentation Manager environments.

THE FUTURE OF PL/I

Future editions of PL/I will include support of SQL and CICS in native mode, SQL, IMS, and CICS in host emulation mode, and object oriented language.



REFERENCES

PL/I Package/2 Fact Sheet (IBM Doc. GC26-3090)

PL/I Package/2 Language Reference (IBM Doc. SC26-4288-00)

PL/I Package/2 Programming Guide (IBM Doc. SC26-4822-00)

PL/I Package/2 Reference Summary (IBM Doc. SX26-3793)

PL/I Package/2 Installation (IBM Doc. SX26-3822)

PL/I Package/2 Language Environment Run-Time Messages (IBM Doc. SC26-3133)

PL/I Package/2 Licensed Program Specifications (IBM Doc. GC26-4821)

PM Programming Reference Volume I (IBM Doc. S10G-6264-00)

PM Programming Reference Volume II (IBM Doc. S10G-6265-00)

OS/2 Presentation Manager Application (IBM Doc. GG24-3422-0)

SAA CPI PL/I Reference (IBM Doc. SC26-4381)

The *Programming Guide*, *Language Reference*, *Reference Summary*, and *Language Environment Run-Time Messages* can be ordered as a set from IBM's Mechanicsburg Distribution Center (IBM Doc. SBOF-3014).

To order a copy of PL/I with companion video for a free trial, call 1-800-426-3346 xSTL5.

ACKNOWLEDGMENTS

Input for this article was provided by members of the PL/I development team. The author would like to thank Peter Elderon and Keson Khieu for their valuable input and reviews.

Davinder S. Athwal, IBM Corp., 555 Bailey Ave., San Jose, Calif., 95141. Athwal has worked for IBM in the U.K. and Canada, dealing with online business systems and language and database products. In 1979, he moved to IBM Santa Teresa and worked on DB2. Since 1983, Athwal has been the PL/I architect responsible for the development of System/370 OS PL/I Version 2, PLITEST, INSPECT, SAA PL/I, and OS/2 PL/I.



- 
- Attend and Receive:
- IBM OS/2
 - Beta version of IBM C++ Set/2
 - IBM LAN Software
 - IBM Communication Manager
 - Lotus Freelance Graphics
 - Computer Associates CA-REALIZER
 - And more ...



The largest IBM OS/2 and LAN Conference ever, for software designers, technical coordinators, LAN experts and administrators, independent programmers, corporate developers, consultants, training executives, and MIS managers

The conference features:
Keynote Speaker

JAMES A. CANNAVINO
IBM Senior Vice President/General Manager
Personal Systems
International Business Machines Corporation

GO WEST!

Explore the Boldest New Computing Frontiers
at the Exciting

OS/2

TECHNICAL INTERCHANGE

Introducing

IBM LAN SYSTEMS

February 28-March 3, 1993

Pointe Hilton at South Mountain
PHOENIX, ARIZONA

OS/2 The POWER of the Future ... NOW

Registration fee is \$595 if received by January 15, 1993.

For Information or to Register, Call OS/2 Technical Interchange Hot Line:

1-800-GET-OS20

(1-800-438-6720 in USA & Canada) • Outside USA & Canada call 1-203-261-6227

IBM and OS/2 are registered trademarks of International Business Machines Corporation. © 1992 IBM Corporation.





GUI

Programming the OS/2 Container Control: The Basics



Peter Haggard

by Peter Haggard and Peter Brightbill

The container control is shipped in OS/2 2.0 as a 32-bit control and in the IBM CUA controls library (CCL/2) as a 16-bit control. The container is functionally equivalent in the two products, and the information in this article can be used by application developers working with either product. This article assumes the reader has limited knowledge of the container control or at least has read the information in the OS/2 2.0 Programming Guide Volume II, Chapter 18. More information on the container control will follow in the next issue.

The container control is large and complex. While this article is not intended to cover every aspect of the container, it provides information that can help developers write an application using it.



Peter Brightbill

STRUCTURE OF A CONTAINER APPLICATION

The container control was designed to be extremely flexible to satisfy the needs of most applications. There are several basic concerns when designing an application. First, applications need not expose all the container's functions, such as drag and drop or the direct edit feature. To prohibit the editing of text in the container, set the `CCS_READONLY` style at creation. The container provides many views to display its data; the application can choose to support some or all of those views.

Developers must also decide how to allocate and insert container records. For the purpose of optimization, records should be allocated and inserted in groups rather than individually. Grouping records reduces the time needed to allocate records and populate

the container. You can allocate and insert n records with two `WinSendMsg` calls, as opposed to $2n$ `WinSendMsg` calls if you were to allocate and insert each separately. A good rule is to avoid any excessive `WinSendMsg` calls. This is especially important with 16-bit `PMWIN` in OS/2. In this setup, the 32-bit application calls 16-bit code in `PMWIN`, which in turn calls the 32-bit container control. On the return, the 32-bit container code returns to the 16-bit `PMWIN` code and then back to the 32-bit application code. Converting between 16- and 32-bit memory models, or "thunking," can be costly in terms of clock cycles.

STRUCTURES

RECORDCORE and MINIRECORDCORE

The container uses one of two structures to store record data, `RECORDCORE` and `MINIRECORDCORE`. The structure used depends on the style set by the application when a container window is created. (The `CCS_MINIRECORDCORE` style indicates that the container is to use `MINIRECORDCORE`.) `MINIRECORDCORE`, which minimizes application memory use, uses 28 bytes, or half the size of the `RECORDCORE` structure's 56 bytes. But the ability to use less memory is rarely free; this situation is no exception. Certain functions are inoperable, or at least more complicated, with the `MINIRECORDCORE` structure.

With `MINIRECORDCORE`, the tree name view cannot be used properly. The bitmaps and icons displayed in this view are stored in the `TREEITEMDESC` structure, which is pointed to by `RECORDCORE`. If you use the tree name view with `MINIRECORDCORE`, the same icon designates the expanded and collapsed icon. This can be confusing to the user. In addition, the

MINIRECORDCORE is restricted to one text string pointer that displays text for each record in text, name, icon, and tree view. RECORDCORE, however, provides a separate text string pointer for each view. In MINIRECORDCORE, you cannot use bitmaps, as only one icon handle is provided (compared to two icon and two bitmap handles in RECORDCORE).

Unlike RECORDCORE, MINIRECORDCORE does not provide a mini icon handle. To use mini icons with MINIRECORDCORE, provide a mini icon handle for the `hptrIcon` field of the MINIRECORDCORE structure. Set the `slBitmapOrIcon` field of the CNRINFO structure to specify icon size. This is necessary because the container is expecting a system default icon size. The container uses the system values `SV_CXICON` and `SV_CYICON` to determine the size of a default icon. If the given icon is a different size than that specified in the `slBitmapOrIcon` field, it will be stretched or compressed as needed. This technique is shown in Figure 1.

After determining which of the two container record storage structures to use, you may want to store more information per record than is provided by either RECORDCORE or MINIRECORDCORE. Determine what information you wish to store for each record, then allocate the extra memory on the `CM_ALLOCRECORD` message. Applications that use the details view usually add more memory to each record to store information to be displayed in the columns. In an object-oriented environment in which the record represents an object, the instance data for the object can be added to the record. This is the technique used by the OS/2 2.0 Workplace Shell, which uses the container control for its folder object.

If any additional data is to be added to each record, the application needs to typedef a structure as shown in Figure 2.

MINIRECORDCORE or RECORDCORE must be the first field of the structure. This is important, as the container always returns pointers to RECORDCORE or MINIRECORDCORE. You can then typecast the pointer to, in this case, PRECORDOBJECT, and be able to address the container record and all additional data. In addition, when an application has to pass a pointer to a record for a particular message, it must point to either MINIRECORDCORE or RECORDCORE, as shown in Figure 3.

CNRINFO and CM_SETCNRINFO

An internal CNRINFO structure describes most of the relevant information used by the container. To change the information in this structure, use the `CM_SETCNRINFO` message. The application should allocate its copy of the CNRINFO structure and set all fields that are to be changed. The



```
/* Using the MINIRECORDCORE to display mini icons in the container. */
PMINIRECORDCORE pRecord;
CNRINFO          CnrInfo;

/* Specify a size for the mini icon. The system value for the size of
 * a menu is commonly used. */
CnrInfo.slBitmapOrIcon.cx = WinQuerySysValue (HWND_DESKTOP, SV_CYMENU);
CnrInfo.slBitmapOrIcon.cy = CnrInfo.slBitmapOrIcon.cx;

/* Tell the container to display all icons this size. */
WinSendMsg (hwndCnr, CM_SETCNRINFO,
            MPFROMP(&CnrInfo), MPFROMLONG(CMA_SLBITMAPORICON));

/* Assign the mini icon to the MINIRECORDCORE. Then insert the
 * record into the container. */
pRecord->hptrIcon = WinLoadPointer (HWND_DESKTOP, NULL, ID_MINIICON);

WinSendMsg (hwndCnr, CM_INSERTRECORD,
            MPFROMP(pRecord), MPFROMP(&RecordInsert));
```

Figure 1: Mini icons with MINIRECORDCORE

```
typedef struct _RECORDOBJECT
{
    MINIRECORDCORE MiniRec; <-- Be sure the MINIRECORDCORE or
    ULONG          ulCount; <----- RECORDCORE is first.
    BOOL           bInit;
    SHORT          sID;        | Additional data.
    PSZ            pszDept;    |
    CDATE          Date;      <-----
} RECORDOBJECT;
typedef RECORDOBJECT *PRECORDOBJECT;
```

Figure 2: Application defined container item

application should send the `CM_SETCNRINFO` message, setting the appropriate flags to indicate which fields are to be updated. The container will then copy the information to the internal CNRINFO structure it manages.



CM_QUERYCNRINFO can be used to retrieve a CNRINFO structure in use by the container. This message is necessary if your application needed information about the container stored in the structure. For example, to determine the container's current view, use CM_QUERYCNRINFO and check the attributes in the flWindowAttr field.

```
PRECORDOBJECT pRecObj;
PRECORDOBJECT pRecParent;
RECORDINSERT RecordInsert;

/* Allocate a MINIRECORDCORE plus the additional bytes needed from the
 * RECORDOBJECT structure. */
pRecObj = (PRECORDOBJECT)WinSendMsg(hwndCnr, CM_ALLOCORECORD,
                                     MPFROMLONG(sizeof(RECORDOBJECT) -
                                                  sizeof(MINIRECORDCORE)),
                                     MPFROMLONG(nRecords));

/* Assign the necessary data to the record. */
pRecObj->MiniRec.cb = sizeof(MINIRECORDCORE);
pRecObj->MiniRec.hptrIcon = hptrCarIcon;
pRecObj->ulCount = 1;
pRecObj->bInit = TRUE;
pRecObj->sID = 100;
strcpy (pRecObj->pszDept, pszDeptTitle);
.
.
RecordInsert.pRecordParent = (PRECORDCORE)pRecParent;
.
.
/* Insert the record */
WinSendMsg (hwndCnr, CM_INSERTRECORD,
            MPFROMP(&pRecObj), MPFROMP(&RecordInsert));
```

Figure 3: Record allocation and insertion

```
/* Fill in the fields of the CNRINFO structure that we want
 * to update */
CnrInfo.xVertSplitbar = 100;
CnrInfo.pFieldInfoLast = pFieldInfoLastLeft;
CnrInfo.pFieldInfoObject = pFieldInfo;

/* Using the proper flags, tell the container to use this
 * new information */
WinSendMsg (hwndCnr, CM_SETCNRINFO, MPFROMP(&CnrInfo),
            MPFROMLONG(CMA_XVERTSPLITBAR | CMA_PFIELDINFOLAST |
                       CMA_PFIELDINFOOBJECT));
```

Figure 4: Updating CNRINFO

THE DETAILS ON DETAILS VIEW

Details view is one of the more popular views, and is the only view that uses the FIELDINFO structure. It can be one of the more complicated views to program to. The details view is a table of rows and columns. Each row is represented by a RECORDCORE or MINIRECORDCORE structure, and each column by a FIELDINFO structure. Many container attributes apply to the entire details view, while others can be set specifically for each column.

Fields from the CNRINFO structure that apply to the entire details view are xVertSplitbar, pFieldInfoLast, and pFieldInfoObject. Respectively, they set the x position of the splitbar, the last column in the left split window, and the column to receive IN-USE (CRA_INUSE) emphasis. Each is optional, depending on the setup of the details view. If your application does not display a splitbar, the xVertSplitbar and pFieldInfoLast fields are not used. The pFieldInfoObject field, if not set specifically, will default to the first column in the left window or the first column in an unsplit details view. All these fields can be specified individually or as a group with the CM_SETCNRINFO message. Figure 4 shows how to set these fields with a single CM_SETCNRINFO message.

Other attributes, the CFA_* attributes, apply only to a specific column in the details view. Some apply only to the titles and others to the data area of the column. Memory for the FIELDINFO structure is allocated with the CM_ALLOCDETAILFIELDINFO message. The flTitle and flData fields of FIELDINFO are used to set specific attributes for each column before they are inserted into the container with CM_INSERTDETAILFIELDINFO. Any attribute can be changed after the FIELDINFOS are inserted by sending the CM_INVALIDATEDDETAILFIELDINFO message.

To display column titles in details view, set the CA_DETAILSVIEWTITLES container attribute in the flWindowAttr field of the CNRINFO structure, then send the CM_SETCNRINFO message specifying CMA_FLWINDOWATTR. For example:

```
CnrInfo.flWindowAttr = CV_DETAIL |
                      CA_DETAILSVIEWTITLES |
                      CA_DRAWICON;
```



```
WinSendMsg (hwndCnr, CM_SETCNRINFO,
            MPFROMP(&CnrInfo),
            MPFROMLONG(CMA_FLWINDOWATTR));
```

When column titles are to be used in the details view, specify the type of data to appear in each column title. This is done through the `f1Title` field of the `FIELDINFO` structure. Only two types of data, first, icons or bitmaps and second, text strings, are allowed in the details view column titles. When `CFA_BITMAPORICON` is specified, the container assumes that the `pTitleData` field points to an icon or bitmap handle, depending on whether the `CA_DRAWICON` or `CA_DRAWBITMAP` attribute is specified. If the `CFA_BITMAPORICON` attribute is not specified, the container will assume that the `pTitleData` field points to a text string.

Column data can be icons, bitmaps, text strings, dates, times, or numbers. These are specified by setting the `f1Data` field of the `FIELDINFO` structure to `CFA_BITMAPORICON`, `CFA_STRING`, `CFA_DATE`, `CFA_TIME`, or `CFA_ULONG`, respectively. When one of these attributes is specified, the container assumes that the same type of data is contained at the location specified by the `FIELDINFO`'s `offstruct` field, as in Figure 5.

Other options specific to the `f1Data` field are:

- **CFA_HORZSEPARATOR**—Draws a horizontal line between the column title and the column data.
- **CFA_SEPARATOR**—Draws a vertical line after a column; the line extends through the title area.
- **CFA_OWNER**—Allows the application to ownerdraw a column.
- **CFA_INVISIBLE**—Makes a column invisible.
- **CFA_FIREADONLY**—Makes a column's data, but not its title, read-only. To make a column title read-only, specify `CFA_FITITLEREADONLY` in the `f1Title` field.

Data in a column can be aligned in any of nine different ways using `CFA_TOP`, `CFA_BOTTOM`, and `CFA_VCENTER` for vertical positioning and `CFA_LEFT`, `CFA_RIGHT`, and `CFA_CENTER` for horizontal positioning. These can be set differently for title and data in one column.

The most important field in the `FIELDINFO` structure is `offstruct`, which tells the column at what offset from the beginning of `RECORDCORE` or `MINIRECORDCORE` to find its data. All data displayed in details view is contained in or after `RECORDCORE` or `MINIRECORDCORE`. If any data is to be contained after, allocate record memory as shown in Figure 3. Figure 5 shows how `FIELDINFO` structures could be set up to display



```
PFIELDINFO pFieldInfo;

if (pFieldInfo = WinSendMsg (hwndCnr, CM_ALLOCDTAILFIELDINFO,
                            MPFROMSHORT(numColumns), NULL))
{
    /* First column title will be a string, and the data will be
     * an icon */
    strcpy (pFieldInfo->pTitleData, pszFirstColTitle);
    pFieldInfo->f1Data = CFA_BITMAPORICON;
    pFieldInfo->offstruct = FIELDOFFSET(RECORDOBJECT, MiniRec.hptrIcon);
    .
    .
    pFieldInfo = pFieldInfo->pNextFieldInfo;

    /* Second column title will be a string, and the data will be
     * the name */
    strcpy (pFieldInfo->pTitleData, pszSecondColTitle);
    pFieldInfo->f1Data = CFA_STRING;
    pFieldInfo->offstruct = FIELDOFFSET(RECORDOBJECT, MiniRec.pszIcon);
    .
    .
    pFieldInfo = pFieldInfo->pNextFieldInfo;

    /* Third column title will be an icon of a calendar, and the data
     * will be the date */
    pFieldInfo->f1Title = CFA_BITMAPORICON;
    pFieldInfo->pTitleData = hptrCalendar;
    pFieldInfo->f1Data = CFA_DATE;
    pFieldInfo->offstruct = FIELDOFFSET(RECORDOBJECT, Date);
    .
    .
}
}
```

Figure 5: Setting up `FIELDINFO`

different columns of data with the `RECORDOBJECT` structure defined earlier. Be sure that if any column contains string data, the corresponding `FIELDINFO`'s `offstruct` parameter must refer to a PSZ (pointer to a character string), not a character string itself.



TREE VIEW

The tree view is used when the objects of a container are best represented in a hierarchy. The drives folder of OS/2 2.0, where a drive's directory structure is displayed, is an example of where a tree view is useful. In all, the container offers three different types of tree views: tree-text, tree-icon, and tree-name.

The layout of the tree view is flexible and can be tailored to an application. The lines that connect child records to their parents can be specified as to their thickness and indentation depth. These values are set with the `cxTreeLine` and `cxTreeIndent` fields of the `CNRINFO` structure, respectively. This example shows how to display longer and thicker lines than those provided by default:

```
CnrInfo.cxTreeLine = 5;
CnrInfo.cxTreeIndent = 50;
WinSendMsg (hwndCnr, CM_SETCNRINFO, &CnrInfo,
            CMA_CXTREELINE | CMA_CXTREEINDENT);
```

The tree-icon and tree-text views use an icon to display the state (expanded or collapsed) of a parent record. The container provides default expanded and collapsed icons, which can be replaced with the `hptrExpanded` and `hptrCollapsed` fields of the `CNRINFO` structure. As with all icons used by the container, these can be sized. The following code sample makes the expanded and collapsed icons larger than the default size.

```
CnrInfo.slTreeBitmapOrIcon.cx = 100;
CnrInfo.slTreeBitmapOrIcon.cy = 100;
WinSendMsg (hwndCnr, CM_SETCNRINFO, &CnrInfo,
            CMA_SLTREEBITMAPORICON);
```

The tree-name view can be used to preserve screen real estate. In this view, the record's expanded and collapsed icon are two separate icons. The container displays the appropriate icon based on the state of a parent record. In this case, the application should utilize the `TREEITEMDESC` structure. Because the `MINIRECORDCORE` structure does not contain a `PTREEITEMDESC` field, the `TREEITEMDESC` structure and tree-name view should be used only with the `RECORDCORE` structure. For all records, the appropriate icons should be loaded into the `TREEITEMDESC` structures `hptrExpanded` and `hptrCollapsed` fields. This code sample loads

icons to be displayed in tree-name view for a parent record.

```
pRecord->pTreeItemDesc = malloc(
    sizeof(TREEITEMDESC) );
pRecord->pTreeItemDesc->hptrExpanded =
    WinLoadPointer(...);
pRecord->pTreeItemDesc->hptrCollapsed =
    WinLoadPointer(...);
```

To insert records in tree view, the `RECORDINSERT` structure needs to be set up properly. To insert records at the root level (items with no parents), the `pRecordParent` field should always be set to `NULL`. There are two options when inserting child records. If inserting the first or last (`CMA_FIRST` or `CMA_END`) child, the `pRecordParent` field must point to the appropriate parent record. When inserting child records that are not first or last, the `pRecordOrder` field must point to the previous child record and the `pRecordParent` field should be set to `NULL`.

Although the container's tree view is flexible, there are some limitations. Regardless of the selection model (extended, multiple, or single) chosen, the tree view is always single selection. Only the root-level items may be shown in the other container views. To show child records in other views, you could use record sharing to share these records with another container. While we have mentioned icons in this section, the container offers the same flexibility for bitmaps when the `CA_DRAWBITMAP` attribute is set.

QUERY MESSAGES AND COORDINATE SYSTEMS

There are a number of query messages provided by the container that allow you to query useful information. `CM_QUERYVIEWPORTRECT`, `CM_QUERYRECORDRECT`, and `CM_QUERYRECORDFROMRECT` are query messages that deal with coordinate systems.

`CM_QUERYVIEWPORTRECT` deals with the workspace coordinates and container window coordinates, while the others work only in window coordinates. The workspace is defined by the extreme positions of the visible records and the workspace coordinate system is stationary. The container window viewport (the viewable section of the container's

workspace) contains a subset of the entire workspace that is moved as the container is scrolled. The viewport does not include scroll bars and titles that may be present in the container window. `CM_QUERYVIEWPORTRECT` can retrieve the viewport's size and position relative to the workspace (`CMA_WORKSPACE`) or relative to the container window (`CMA_WINDOW`). If only the size of the viewport is desired, either coordinate system can be used.

`CM_QUERYRECORDRECT` is used to query a record's position relative to the container window's origin. This value is dependent on the viewport's current location and changes when the container is scrolled. The position of the record in workspace coordinates can be queried with `CM_QUERYRECORDINFO`. The record's workspace position does not change when the container is scrolled.

It may be useful to determine which records intersect a given rectangle whose coordinates are relative to the container window origin, for example, if you wanted to know which record was under the mouse. With the mouse coordinates expressed as a rectangle, use `CM_QUERYRECORDFROMRECT` to search for the record under the mouse, if one exists. This code sample can be used in any view to retrieve a record under the mouse:

```
QueryRecordFromRect.cb =
    sizeof(QUERYRECFROMRECT);
QueryRecordFromRect.rect.xLeft = MousePt.x;
QueryRecordFromRect.rect.xRight = MousePt.x;
QueryRecordFromRect.rect.yTop = MousePt.y;
QueryRecordFromRect.rect.yBottom = MousePt.y;
QueryRecordFromRect.rect.fsSearch = CMA_PARTIAL
    | CMA_ITEMORDER;
pRecord = WinSendMsg (hwndCnr,
    CM_QUERYRECORDFROMRECT,
    MPFROMP(CMA_FIRST),
    &QueryRecordFromRect);
```

You may want to use the `CMA_ZORDER` flag in the `fsSearch` field if the container is in a nonautopositioned icon view (one in which the `CCS_AUTOPOSITION` is not set upon container creation). In this case, records may be positioned on top of one another.

CLOSING A CONTAINER APPLICATION

When your application is closed, it is responsible for freeing all memory, such as memory allocated for text strings used in records and columns. All icons and bitmaps loaded should also be released.

Peter Brightbill, IBM Programming Systems Laboratory, 11000 Regency Pkwy., Cary, N.C., 27512. Brightbill is a senior associate programmer in OS/2 PM extensions development. He joined IBM in 1989 and has been working in OS/2 PM development since that time. Brightbill was a member of the OS/2 container control development team and is currently working on additional OS/2 controls. He received a B.S. in mathematics and computer science from Millersville University, Pa.

Peter Haggar, IBM Programming Systems Laboratory, 11000 Regency Pkwy., Cary, N.C., 27512. Haggar is a staff programmer in OS/2 PM extensions development. He joined IBM in 1987 and has been working in OS/2 PM development since 1989. Haggar was a member of the OS/2 container control development team and recently completed a five-month assignment with the OS/2 2.0 Workplace Shell development team. He received a B.S. in computer science from Clarkson University, N.Y.



REFERENCES

Haggar, Peter, Tai Woo Nam, and Ruth Anne Taylor. "Container Control: Implementing the Workplace Model," *IBM Personal Systems Developer* (Winter 1992): 48-54.

OS/2 2.0 Programmer's Guide Volume II (IBM Doc. S10G-6494)

OS/2 2.0 Presentation Manager Programming Reference Volume III (IBM Doc. S10G-6272)

SAA CUA Guide to User Interface Design (IBM Doc. SC34-4289)

SAA CUA Advanced Interface Design Reference (IBM Doc. SC34-4290)

GUI

Implementing a Wastebasket In the Workplace Shell



Dave Hock

by David J. Hock

OS/2 2.0's shredder device is useful for permanently deleting items. To use the shredder, the user drags files or other items that are no longer wanted to the shredder icon on the desktop, and the item is instantly discarded. The main shortcoming of the shredder is that a shred operation is not very forgiving. In contrast, the wastebasket is a delete device from which users can recover previously discarded items.

With the wastebasket, unwanted items are dragged to an icon on the desktop. Later, you can open the wastebasket and retrieve items placed there. When you are ready to permanently discard its contents, empty the wastebasket using a pop-up menu. You can also select an auto-empty feature so the wastebasket is automatically emptied each time you start your system.

To implement the wastebasket, you must modify some behaviors of the folder class.

The wastebasket device, shown in Figure 1, can be easily added to the OS/2 2.0 Workplace Shell, using existing object-oriented code as a model. This article explains the general object-oriented programming principles that are used to create a new object such as the wastebasket. It then describes how the wastebasket device was implemented in the OS/2 2.0 Workplace Shell.

THE WORKPLACE SHELL AND OBJECT-ORIENTED PROGRAMMING

Although the OS/2 2.0 Workplace Shell was developed in C, the System Object Model (SOM) environment allowed the Workplace Shell to support object-oriented programming, which uses objects consisting of data and functions that operate on that data. This

grouping of data and function is known as encapsulation. Encapsulation is an excellent technique for hiding data and protecting it from accidental corruption. Together, the function and the values of the data define the behaviors of an object.

An object can be thought of as a group of characteristics or an object class. An object class has one parent and many children that adopt behaviors from their parent object. Programmers can create new object types by creating descendant object classes (subclasses), reusing all parent behaviors needed for the new object class, and then modifying any behaviors that do not fulfill the requirements of the new object class. New behaviors can also be added to the new object class.

A new object class can be very similar to its parent or it can be radically different, with little reuse of the parent behaviors. Since a very different child class requires extensive modifications to the new object class, it is desirable to choose a parent class that has as much as possible in common with the required object class.

Applying Object-Oriented Programming to the Wastebasket

To create a wastebasket, we need to find a similar object to use as a parent class. At first, a shredder seems the most likely candidate; both devices are used to delete items. But a shredder does not allow you to view its contents or remove items. In fact, a shredder exhibits no containment behavior at all. A folder, however, exhibits many behaviors needed in a wastebasket. You can drop items on a folder icon and view and remove its contents. Since a folder is behaviorally closest

to a wastebasket, it was selected as a parent class for the wastebasket.

To implement the wastebasket, you must modify some behaviors of the folder class. A "cannot drop" icon should be displayed if undeletable items are dragged over the wastebasket. The wastebasket icon and default title differ from those of a folder; as the wastebasket should not be copied, it should not have a Copy command in its pop-up menu. Instead, add an Empty now command to the menu and an auto-empty page to the settings view notebook. Also remove the Include page from the notebook, so users cannot hide the wastebasket contents.

USING THE FOLDER TO IMPLEMENT THE WASTEBASKET

Using an existing object class to create a new object class is known as "subclassing." Once an object class is subclassed, existing behaviors are modified and new ones are added by writing C functions called "methods."

Each class has an associated abstract class object, the factory object, that creates instances of a class. The methods of a class object, or class methods, are defined along with instance methods when defining a class.

Class methods operate on the class as a whole, allowing memory and methods to be efficiently shared by an entire class of objects. Icons, for example, are bitmap images that require a fair amount of memory. Storing an icon with its class rather than with each instance or occurrence of an object saves memory.

You can also save time by running a class method once for the entire class rather than for each instance of a class. Similarly, the code that loads an icon into memory is run only once during initialization of the class.

In Figure 2, subclassing is done by creating a class definition file for the wastebasket. This file consists of several parts that define the object and its methods:

- Class definition
- Parent definition

- Passthrough data
- New methods
- Modified or overridden methods

The class definition specifies the name of the object class, its version number, and the prefix to be used for method names. The parent definition specifies the parent for the wastebasket class. In this case, `WPFolder`, the folder class, will pass on its behaviors and methods to the wastebasket class.

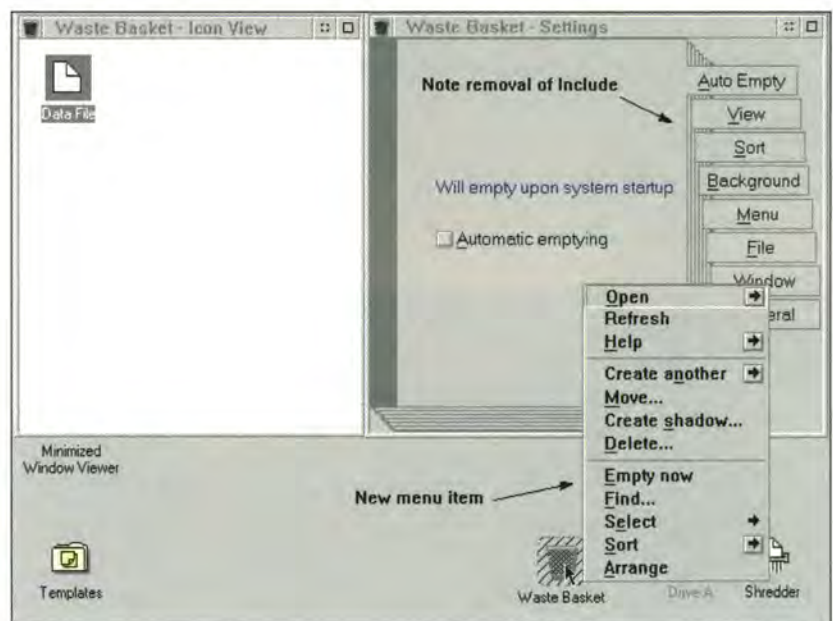


Figure 1: The wastebasket

The passthrough section passes C definitions through to the include file, where they will be interpreted by the C compiler. In this example, the constant `IDKEY_AUTOEMPTY` is defined as an index for saving and retrieving the value of the object data.

Instance data, or object occurrence data, defines the state of an object. In this example, the instance data is very simple. A Boolean variable is declared to store the auto-empty flag. If the flag is set to true, the wastebasket will be emptied at system startup.

Methods usually have similar parameters; the first parameter is a pointer used to reference the object instance and call other methods from within the method.





For direct access to instance data, call the `GetData` method at the beginning of a method. The method name will have a prefix that corresponds to its class, as in `WPWastBGetData`. It

```
# WASTEBSK.CSC          by Dave Hock

# Include the class definition file for the parent class

include <wpfolder.sc>
# Define the new class
class: WPWastB,
    file stem = wastebsk,
    external prefix = wpwstb_,
    class prefix = wpwstbcls_,
    major version = 1,
    minor version = 1,
    local;
# Specify the parent class
parent: WPFolder;
# Specify the release order of new methods
release order: SetAutoEmptyFlag, QueryAutoEmptyFlag;
# passthru to C compiler
passthru: C.ih;
#define IDKEY_AUTOEMPTY 9000
endpassthru;
# Define instance data for the class
data:
    BOOL bAutoEmpty;
#
# Define new methods
methods:
    BOOL QueryAutoEmptyFlag(), private;
    BOOL SetAutoEmptyFlag(BOOL bAutoEmpty), private;
# Specify methods being overridden
override wpInitData;
override wpModifyPopupMenu;
override wpMenuItemSelected;
override wpSetup;
override wpSaveState;
override wpRestoreState;
override wpQueryStyle;
override wpDragOver;
override wpAddFolderIncludePage;
override wpAddSettingsPages;
override wpclsQueryTitle, classmethod;
override wpclsInitData, classmethod;
override wpclsQueryIcon, classmethod;
override wpclsUnInitData, classmethod;
```

is better to access instance data by calling access methods, rather than referencing the variable directly. This allows the programmer to subclass an object and override the instance data access methods. Data access methods must, however, directly access the data being managed. In our example, two new methods that access the instance data item for the auto-empty flag are declared for the wastebasket object: `QueryAutoEmptyFlag` and `SetAutoEmptyFlag`.

Finally, declare any existing folder methods that will be modified or overridden for the wastebasket. These methods dictate how folder behavior will be modified to create the wastebasket. The code for the wastebasket methods is contained in the `.C` file.

OVERRIDING FOLDER BEHAVIORS

Most of the creation of the wastebasket consists of overriding parent methods, both class and instance.

When overriding a parent method, it is important that the parent method and functions be called within the new method, or important behaviors may be lost. Parent methods are called with a `parent_` method prefix. For example, even though custom code is executed in the `WpMenuItemSelected` when a user selects "Empty now" in the pop-up menu, the parent method for menu selection must still be called to allow existing folder menu items to function.

The following wastebasket class methods are overridden in C to create the desired behaviors:

- **wpclsInitData**—This method is run once during a class initialization to initialize any data required for the class as a whole. With the wastebasket, it is used to load the device's resource data and icons into memory from the resource file, as in Figure 4.
- **wpclsUnInitData**—This method is run when a class is destroyed. It destroys a previously loaded icon, freeing some memory. The resource module is also released from memory.

Figure 2: The class definition file `Wastebsk.CSC`



```

/*****
 * METHOD:  wpclsQueryTitle
 * PURPOSE: This class method returns the default folder title for any instance
 *           of the password protected folder class. This title is used if a title is
 *           not supplied in the WinCreateObject() call.
 * INVOKED: By the Workplace Shell, upon instantiation of the object.
 *****/
SOM_Scope PSZ SOMLINK wpwstbcls_wpclsQueryTitle(M_WPWastB *somSelf) {
    return("Waste Basket);          /* Return default title */
}

/*****
 * METHOD:  wpclsInitData
 * PURPOSE: This class method allows the initialization of any class data
 *           items. The overridden method simply obtains a module handle
 *           to be used when accessing Presentation Manager resources, then
 *           invokes the parent's default processing.
 * INVOKED: By the Workplace Shell, upon loading the class DLL.
 *****/
SOM_Scope void SOMLINK wpwstbcls_wpclsInitData(M_WPWastB *somSelf) {
    CHAR ErrorBuffer[100];
    DosLoadModule((PSZ) ErrorBuffer,          /* Obtain DLL module handle*/
                  sizeof(ErrorBuffer),
                  "WASTEBSK",                 /* Module name */
                  &hmodThisClass);           /* Module handle */
    hIcon=WinLoadPointer(HWND_DESKTOP,        /* Load icons */
                        hmodThisClass, ID_WASTEICO);
    parent_wpclsInitData(somSelf);
}

/*****
 * METHOD:  wpclsQueryIcon
 * PURPOSE: This class method returns the handle to the default icon for
 *           the class. Allows setting of the icon to a wastebasket.
 * INVOKED: By the Workplace Shell, upon instantiation of the object.
 *****/
SOM_Scope HPOINTER SOMLINK wpwstbcls_wpclsQueryIcon(M_WPWastB *somSelf)
{ return (hIcon); }

/*****
 * METHOD:  wpclsUnInitData
 * PURPOSE: This class method allows the release of any class data items
 *           or resources. The overridden method releases the module handle
 *           obtained by wpclsInitData, then invokes the parent's method.
 * INVOKED: By the Workplace Shell, upon unloading the class DLL.
 *****/
SOM_Scope void SOMLINK wpwstbcls_wpclsUnInitData(M_WPWastB *somSelf) {
    WinDestroyPointer(hIcon);
    DosFreeModule(hmodThisClass);             /* Free module handle*/
    parent_wpclsUnInitData(somSelf);
}

```

Figure 3: Class method implementations



Most of the creation of the wastebasket consists of overriding parent methods, both class and instance.

- **wpClsQueryTitle**—This method is called by the Workplace Shell to query the default title for an object class. Overriding this method gives the wastebasket a default title other than “Folder.”

- **wpClsQueryIcon**—This method is called by the Workplace Shell to query an icon used for a class of objects. Overriding this method replaces the folder icon with a custom wastebasket icon.

Many instance methods are overridden to implant the wastebasket’s behavior. If these methods are intercepted, the management of the auto-empty instance variable for the wastebasket can be implemented. The wastebasket’s drag-and-drop behavior, pop-up menu, and settings notebook can also be customized, as in Figure 4.

- **wpInitData**—Overriding this method allows objects to initialize their instance data. In this case, the auto-empty flag is initialized to false, disabling the automatic emptying feature.

- **wpSetup**—This method allows the wastebasket to support a setup string, specified at object creation to initialize objects. A setup string can be specified using the Class Hierarchy Browser in the OS/2 2.0 Toolkit. The \OS2\INI.RC file shows examples of the desktop objects and corresponding setup strings.

The wastebasket supports a setup string that sets the initial value of the auto-empty flag using `AUTOEMPTY=TRUE` or `AUTOEMPTY=FALSE`.

- **wpModifyPopupMenu**—This method allows “Empty now” to be added to the pop-up menu by calling the `_wpInsertPopupMenuItems` method.

- **wpMenuItemSelected**—This method verifies that “Empty now” has been selected. If it has, the folder method `_wpDeleteContents` is called to delete all items in the wastebasket. If a different menu item is selected, the parent method is called so the folder class may process the menu selection.

- **wpSaveState**—This method is called when the Workplace Shell makes an object dormant, during shutdown or when the folder containing the object has been closed for a long time. All instance data introduced by the class can be saved within this method. In the case of

the wastebasket, the value of the auto-empty flag is saved using the `wpSaveData` method and the index `IDKEY_AUTOEMPTY`.

- **wpRestoreState**—This method is called by the Workplace Shell when a dormant object is awakened, restoring the instance data to its previous state. `wpRestoreState` makes the auto-empty flag persistent between system boots.

- **wpQueryStyle**—The wastebasket overrides this method to modify style flags for an object. The `OBJSTYLE_NOCOPY` flag is or’ed to keep the wastebasket from being copied, to prevent “Copy...” from appearing in the pop-up menu, and to disallow a drag-and-copy operation. You could also prevent the wastebasket from being deleted by or’ing the `OBJSTYLE_NODELETE` flag. (The OS/2 2.0 PM Programming Reference section lists all available style flags.)

- **wpDragOver**—This method is one of the most important for the wastebasket; it emulates the shredder, allowing users to drop only deletable objects. If an undeletable object such as System Settings is dragged to the wastebasket, the “cannot drop” icon is displayed. The style of all items being dragged must be checked to make sure they don’t have the `OBJSTYLE_NODELETE` flag setting.

- **wpDrop**—This method is called when items are dropped on the wastebasket. Although it is not done in the example, `wpDrop` could be overridden to put up a message box prompting the user to continue or cancel the drop operation.

- **wpAddFolderIncludePage**—This method is overridden to delete the Include page from the settings notebook. This is the one method in which the parent method is not called, keeping the code that inserts the Include page from running.

- **wpAddSettingsPages**—This method adds an auto-empty page to the settings notebook, first filling in a `PAGEINFO` structure that includes information about the auto-empty dialog. The `pCreateParams` field is passed to the dialog procedure upon initialization. By passing the wastebasket handle, the dialog can reference the wastebasket, call the `Query/SaveAutoEmptyFlag` methods, and manage the value of the checkbox, as in Figure 5.



```

/*****
 * METHOD:   QueryAutoEmptyFlag
 * PURPOSE: Gets the instance data and returns the flag
 * INVOKED: From DlgProc
 *****/
SOM_Scope BOOL SOMLINK wpwstb_QueryAutoEmptyFlag(WPWastB *somSelf) {
    WPWastBData *somThis = WPWastBGetData(somSelf); /* get data */
    return _bAutoEmpty;
}

/*****
 * METHOD:   SetAutoEmptyFlag
 * PURPOSE: Gets the instance data and sets the flag
 * INVOKED: From DlgProc
 *****/
SOM_Scope BOOL SOMLINK wpwstb_SetAutoEmptyFlag(WPWastB *somSelf,
                                                BOOL bAutoEmpty) {
    WPWastBData *somThis = WPWastBGetData(somSelf);
    _bAutoEmpty = bAutoEmpty;
    return (BOOL) 0;
}

/*****
 * METHOD:   wpInitData
 * PURPOSE: Initializes instance data
 * INVOKED: By Workplace Shell, upon instantiation of the object instance.
 *****/
SOM_Scope void SOMLINK wpwstb_wpInitData(WPWastB *somSelf) {
    parent_wpInitData(somSelf);
    _SetAutoEmptyFlag(somSelf, FALSE);
}

/*****
 * METHOD:   wpModifyPopupMenu
 * PURPOSE: Adds an additional "Empty now" item to the object's context menu.
 * INVOKED: By Workplace Shell, upon display of the popup.
 *****/
SOM_Scope BOOL SOMLINK wpwstb_wpModifyPopupMenu(WPWastB *somSelf,
                                                  HWND hwndMenu, HWND hwndCnr, ULONG iPosition) {
    _wpInsertPopupMenuItems(somSelf,          /* Insert menu item */
                            hwndMenu,         /* Menu handle */
                            iPosition,        /* Default position */
                            hmodThisClass,    /* Module handle */
                            ID_CXTMENU_WASTE, /* Menu item identifier */
                            0);               /* No submenu identifier */
    return (parent_wpModifyPopupMenu(somSelf, hwndMenu, hwndCnr, iPosition));
}

/*****
 * METHOD:   wpMenuItemSelected
 * PURPOSE: Processes the user's selections from the context menu. The
 *          method processes only the added "Empty now" item, before
 *          invoking the parent's default processing to handle other items.
 * INVOKED: By Workplace Shell, upon selection of a menu item by the user.
 *****/

```

Figure 4: Instance method implementations (continued on page 108)



```

SOM_Scope BOOL SOMLINK wpwstb_wpMenuItemSelected(WPwastB *somSelf,
    HWND hwndFrame, ULONG ulMenuId) {
    if (ulMenuId==IDM_EMPTY) /* if our new item is selected, do it. */
        return(_wpDeleteContents(somSelf, TRUE));
    else return (parent_wpMenuItemSelected(somSelf,hwndFrame,ulMenuId));
}
/*****
* METHOD    wpSetup
* PURPOSE: Sets folder properties based upon a setup string passed by the
*           object's creator as part of the WinCreateObject() call. The
*           overridden method simply processes the AUTOEMPTY keyword to set
*           the folder's autoempty status and period, before invoking
*           the parent's default processing to handle all other keywords.
* INVOKED: By the Workplace Shell, upon instantiation of the object.
*****/
SOM_Scope BOOL SOMLINK wpwstb_wpSetup(WPwastB *somSelf, PSZ pszSetupString) {
    BOOL      rc;
    CHAR      szValue[256];
    ULONG     cbValue=256;
    rc = parent_wpSetup(somSelf,pszSetupString);
    if(_wpScanSetupString(somSelf,pszSetupString,(PSZ)"AUTOEMPTY",
        szValue,&cbValue))
        if (!strcmp(szValue,"TRUE")) _SetAutoEmptyFlag(somSelf, TRUE);
        else _SetAutoEmptyFlag(somSelf, FALSE);
    return rc;
}
/*****
* METHOD:    wpSaveState
* PURPOSE: Saves the object instance's persistent state data. The
*           overridden method simply saves the autoempty info , then invokes
*           the parent's default processing to handle any other instance
*           data defined by ancestor classes.
* INVOKED: By the Workplace Shell, when the object becomes dormant.
*****/
SOM_Scope BOOL SOMLINK wpwstb_wpSaveState(WPwastB *somSelf) {
    BOOL rc;
    BOOL bAutoEmpty;
    rc = parent_wpSaveState(somSelf);
    bAutoEmpty = _QueryAutoEmptyFlag(somSelf);
    _wpSaveData(somSelf,_somGetClassName(somSelf),IDKEY_AUTOEMPTY,
        (PBYTE)&bAutoEmpty,sizeof(BOOL));
    return rc;
}
/*****
* METHOD:    wpRestoreState
* PURPOSE: Restores the object instance's persistent state data. The
*           overridden method simply restores the autoempty data, then
*           invokes the parent's default processing to handle any other
*           instance data defined by ancestor classes.
* INVOKED: By the Workplace Shell, when the object becomes awake.
*****/

```

Figure 4: Instance method implementations (continued on page 109)



```

SOM_Scope BOOL SOMLINK wpwstb_wpRestoreState(WPwastB *somSelf,
        ULONG ulReserved) {
    BOOL rc;
    BOOL bAutoEmpty;
    ULONG cbValue = sizeof(BOOL);
    rc = parent_wpRestoreState(somSelf, ulReserved);
    if (_wpRestoreData(somSelf, _somGetClassName(somSelf),
        IDKEY_AUTOEMPTY, (PBYTE)&bAutoEmpty, &cbValue) ) {
        _SetAutoEmptyFlag(somSelf, bAutoEmpty);
        if (bAutoEmpty) _wpDeleteContents(somSelf, TRUE);
    } /* endif */
    return rc;
}

/*****
 * METHOD: wpQueryStyle
 * PURPOSE: Allows us to force a style such as OBJSTYLE_NOCOPY or NODELETE
 * INVOKED: By the Shell, to when it wants to copy the object
 *****/
SOM_Scope ULONG SOMLINK wpwstb_wpQueryStyle(WPwastB *somSelf) {
    return (parent_wpQueryStyle(somSelf) | OBJSTYLE_NOCOPY);
}

/*****
 * METHOD: wpDragOver
 * PURPOSE: To detect if the user is trying to drag undeletable objects to
 *           the wastebasket. If they are, we will put up the no-drop sign.
 * INVOKED: By the Shell, when object(s) are dragged over the wastebasket.
 *****/
SOM_Scope MRESULT SOMLINK wpwstb_wpDragOver(WPwastB *somSelf,
        HWND hwndCnr, PDRAGINFO pdrgInfo) {
    MRESULT mResult;
    USHORT usItem;
    SOMany * Object;
    PDRAITEM pDragItem;
    /*Get parent call to figure out if it is droppable.*/
    mResult = parent_wpDragOver(somSelf, hwndCnr, pdrgInfo);
    /*Parent says NEVERDROP then return */
    if (SHORT1FROMMR(mResult) == DOR_NEVERDROP)
        return(mResult);
    /*Check if all items are acceptable */
    for (usItem=0; usItem < pdrgInfo->cditem; usItem++) {
        if ( !(pDragItem = DrgQueryDragitemPtr(pdrgInfo, usItem)) )
            return(MPFROM2SHORT(DOR_NEVERDROP, DO_DEFAULT));
        if ( !(pDragItem -> ulItemID) ) /*No SOM Object ID*/
            return(MPFROM2SHORT(DOR_NEVERDROP, DO_DEFAULT));
        if ( !(Object=OBJECT_FROM_PREC(pDragItem->ulItemID)) ) /*Get ObjectID */
            return(MPFROM2SHORT(DOR_NEVERDROP, DO_DEFAULT));
        if (_wpQueryStyle(Object) & OBJSTYLE_NODELETE) /*Check if deletable*/
            return(MPFROM2SHORT(DOR_NEVERDROP, DO_DEFAULT));
        if (pdrgInfo -> usOperation == DO_COPY || /*Check no Link or Copy*/
            pdrgInfo -> usOperation == DO_LINK) /*If so then return no */
            return(MPFROM2SHORT(DOR_NEVERDROP, DO_DEFAULT));
    }
    /*Loop through all items*/
}

```

Figure 4: Instance method implementations (continued on page 110)



```

        return (MPFROM2SHORT(DOR_DROP,DO_MOVE));
    }
/*****
 * METHOD: wpAddFolderIncludePage
 * PURPOSE: Allows us to remove the include page so the user can't hide any
 *           items in the waste basket. It is important to have all items
 *           visible so that something important is not accidentally emptied.
 * INVOKED: By the Shell, when the page is added to the settings view.
 *****/
SOM_Scope ULONG SOMLINK wpwstb_wpAddFolderIncludePage(WPWastB *somSelf,
        HWND hwndNotebook) {
    WPWastBData *somThis = WPWastBGetData(somSelf);
    WPWastBMethodDebug("WPWastB", "wpwstb_wpAddFolderIncludePage");
    return TRUE;
}

/*****
 * METHOD: wpAddSettingsPages
 * PURPOSE: Allows us to add the AutoEmpty page to the settings view.
 * INVOKED: By the Workplace Shell, when the settings view is opened.
 *****/
SOM_Scope BOOL SOMLINK wpwstb_wpAddSettingsPages(WPWastB *somSelf,
        HWND hwndNotebook) {
    PAGEINFO pageinfo;
    if (!parent_wpAddSettingsPages(somSelf, hwndNotebook))
        return (FALSE);
    memset(&pageinfo, sizeof(PAGEINFO), 0); /* clear the struct */
    pageinfo.cb = sizeof(PAGEINFO);
    pageinfo.pfnwp =DlgProc;
    pageinfo.resid = hmodThisClass;
    pageinfo.pCreateParams = (PVOID)somSelf;
    pageinfo.dlgid = ID_DLG_AUTOEMPTY;
    pageinfo.usPageStyleFlags = BKA_MAJOR;
    pageinfo.usPageInsertFlags = BKA_FIRST;
    pageinfo.pszName = "Auto Empty";
    return _wpInsertSettingsPage(somSelf, hwndNotebook, &pageinfo);
}

```

Figure 4: Instance method implementations (continued from page 109)

CONCLUSION

The wastebasket example is a simple one; it could be extended to add other features such as automatic emptying after a set amount of time or deletion of only certain types of files. The power of the Workplace Shell object-oriented environment enables you to create many custom objects and to fine-tune their properties without repetitive programming.

ACKNOWLEDGMENTS

The author would like to thank editor Judy Hock; Chris Andrew, James Taylor, and the rest of the Workplace Shell Team; the CUA architecture team; Richard Redpath; and the ITSC Boca Raton team.



```

/*****
 * PROCEDURE NAME: DlgProc
 * description: Dialog procedure for auto-empty page dialog
 *****/
MRESULT EXPENTRY DlgProc(HWND hwndDlg, ULONG msg, MPARAM mp1, MPARAM mp2){
    SOMany * Wastebasket;
    BOOL bAutoEmpty;
    switch (msg) {
        /* Determine message class */
        case WM_INITDLG:
            /* Dialog being initialized */
            /* Store SOM pointer in window word QWL_USER */
            WinSetWindowULong(hwndDlg, QWL_USER, (ULONG)LONGFROMMP(mp2));
            Wastebasket = (WPWastB *) LONGFROMMP(mp2);
            if (_QueryAutoEmptyFlag(Wastebasket))
                WinCheckButton(hwndDlg, ID_CB_AUTO, 1);
            break;
        case WM_CONTROL:
            Wastebasket = (WPWastB *)WinQueryWindowULong(hwndDlg, QWL_USER);
            if (SHORT1FROMMP(mp1)==ID_CB_AUTO)
                && (SHORT2FROMMP(mp1)==BN_CLICKED))
                if (WinQueryButtonCheckstate(hwndDlg, ID_CB_AUTO))
                    _SetAutoEmptyFlag(Wastebasket, FALSE);
                else _SetAutoEmptyFlag(Wastebasket, TRUE);
                _wpSaveDeferred(Wastebasket);
            break;
    }
    return(WinDefDlgProc(hwndDlg, msg, mp1, mp2));
}

```

Figure 5: Notebook page dialog procedure

David J. Hock, UCANDU Software, Inc., 2309 Kelly Road, Apex, NC 27502, (919) 387-7391. Hock is a former member of the IBM Advanced CUA Architecture group. He also completed a six-month assignment for the IBM OS/2 2.0 Workplace Shell programming team in Boca Raton, Florida. Hock holds a B.S. in computer science and electrical engineering from the State University of New York, Stony Brook, and an M.S. in management and computer engineering from North Carolina State University.

REFERENCES

OS/2 2.0 Programming Reference toolkit

OS/2 2.0 Programming Guide toolkit

Systems Application Architecture Common User Access Advanced Interface Design Reference (IBM Doc. SC34-4290???)

Systems Application Architecture Common User Access Advanced Interface Design Guide



GUI

Thinking in Presentation Manager



Diane Lovelett

by Christine Grau and Diane Lovelett

This article describes some of the differences we noticed while coding host and workstation interfaces. Our experience is primarily with Presentation Manager (PM) on the workstation and the Interactive System Productivity Facility (ISPF) and Dialog Tag Language (DTL) on multiple virtual storage (MVS).

There are several reasons why you might port a host application and connect to a workstation. The main reason is that a workstation interface is a graphical interface that provides flexibility and capability through a wide choice of fonts, numerous colors, and a wide variety of controls that allow programmers to improve an application's overall usability.

A workstation application changes dynamically, allowing extensive interface tailoring. This feature allows programmers to create precise interfaces that communicate to the user exactly the task at hand.

Workstation applications increase user response times, especially in the area of screen painting. For cooperative processing applications, workstation applications can, in some cases, offload some of the work from the host, saving millions of instructions per second (MIPS), leading to actual dollar savings for the organization as a whole.

The applications are usually object-oriented and conform to the messy desk and workplace models. Additionally, OS/2 provides multitasking, which allows for improvements in performance and enables multiple copies of an application to run at the same time.

Whatever motivates your decision to port your application from the host to the workstation, there must be a shift in thinking for the developer used to coding a host interface. The developer must understand the differences between the nature of each interface and the differences between the techniques used to create each interface.

GRAPHICAL VS. NONGRAPHICAL INTERFACES

A PM interface provides a graphical environment. A traditional program gets user input from the keyboard and displays output to the screen.

As stated by Charles Petzold in *Programming the OS/2 Presentation Manager* (Microsoft Press, 1989):

...a graphical windowing environment can inspire program developers to create a radically new and environment of Presentation Manager is rich in function—programs can use graphics and formatted text to convey a high density of information to the user.

...With the addition of a mouse, the screen becomes a potential source of user input. Logic within Presentation Manager assists the application in obtaining user input from various controls on the screen, such as menus, scroll bars, buttons, and dialogue boxes. The interaction between the mouse and screen narrows the gap between user and program.

Because a PM interface is a graphical interface, it can be thought of as three-dimensional, even when the interface contains traditional controls

There must be a shift in thinking for the developer used to coding a host interface.

and not graphics per se. A host interface can be thought of as two-dimensional. What you see when you first look at a Presentation Manager interface is not the whole picture, while what you see when you look at a host interface is the whole picture. For example, a PM drop-down combo box looks compact and graphically small. However, when you click on the drop-down button, a new view is seen. Suddenly, an entire list of items is visible, which can once again be hidden with another click. This all occurs without leaving the main window.

As a result, you think differently when designing a PM interface than when designing a host interface. For example, you can put a lot of data into a single window without crowding the window. If you first designed a window on Presentation Manager and tried to design a comparable window on the host, it would be difficult because the host cannot provide the function in the same way, cannot display as much data in a single panel, and must use less flexible techniques to achieve similar results. The results may be similar, but never the same.

Conversely, you can redesign a host interface into a PM interface without limitations and with an increase in function. Additionally, a PM interface allows you a greater variety of font styles and sizes, colors, and extensive graphics.

Designing a PM interface is a more intriguing and difficult task than designing a host interface because the tools available for designing a PM interface are varied and extensive. Because the tools for designing a host interface are limited, there are fewer design decisions to be made and interface design is easier and less time-consuming.

A MESSY DESK VS. A HIERARCHICAL ENVIRONMENT

A PM interface gives users a messy desk environment in which to work. A host interface provides a hierarchical environment to users. The PM interface also allows multitasking. Switching between applications or within an application can be done with a keystroke or a click of the mouse. The host, on the other hand, often forces you to exit the

application you are in before beginning another task.

When programming for the host, programmers must not only do extensive task analyses to determine what tasks their users will do, they must also understand in what order they will do these tasks. The finished application must allow users to perform certain tasks in a specific order.



PM TERMS	HOST TERMS
Check box	
Client window	
Combo box	Selection list
Container	
Dialogue window	Pop-up
Entry field	Entry field
Frame window	Panel
Group box	
Icon	
List box	Selection list
Menu	Menu
Message box	Pop-up
Multiple line entry	Edit session
Notebook	
Pushbutton	Program function (PF) keys, enter key, or commands
Radio button	
Scroll bar	PF8 or PF7 (or other defined PF keys)
Slider bar	
Spin button	
Static text	Static text
System menu	
Value set control	

Table 1: Presentation Manager terms vs. host terms

A workstation interface allows users to choose when they will do certain tasks; they are not tied to the hierarchy of the program.

INTERFACE ELEMENTS

PM interface elements, such as windows and controls, offer more selection and functions than their counterparts on the host, such as panels, program function (PF) keys, and pop-



ups. The host screen and workstation window are comparable entities. Table 1 compares PM and host interface elements to allow you to see relationships among the various elements.

```
/* Message Loop */

while (WinGetMsg(hab, (PQMSG)&qmsg, (HWND)NULL, 0, 0))
    WinDispatchMsg(hab, (PQMSG)&qmsg);
```

Figure 1: The Presentation Manager message loop

```
MRESULT EXPENTRY WinProc(HWND hwnd, USHORT msg, MPARAM
mp1, MPARAM mp2)
{
    switch (msg)
    {
        case WM_CREATE :

            /* code to create window and initialize data */

            break;
        case WM_PAINT :

            /* code to paint the screen */

            break;
        case WM_SIZE :

            /* code to when a window is resized */

            break;
        case WM_DESTROY :

            /* code to destroy a window and free resources */

            break;
        default :
            return (WinDefWindowProc(hwnd, msg, mp1, mp2));
    }
    return (FALSE);
}
```

Figure 2: Skeleton window procedure

The large number of PM interface elements allows programmers to be more creative in displaying, retrieving, and connecting data.

Interface elements are static. PM controls can dynamically change based on something chosen by the user. An entire PM window can be moved or resized, something that is impossible on a host screen.

On the host, accelerated paths to functions can be provided with a command language or through a set of 24 PF keys. However, the workstation can provide many more functions with accelerator keys.

An additional drawback to the host command language mechanism of providing fast paths is that these commands are not illustrated next to the menu items they represent. The learning required to remember the commands is separate from becoming familiar with the interface. With PM, the accelerator key for a function is visible with the function.

The PM drag-and-drop function provides users with direct manipulation capabilities. With a drag-and-drop function, a user can drag a file, which is represented as an object, to an icon of a printer to print the file. On the host, users must type commands or follow an application's path to print a file and cannot directly manipulate the file object.

Although not specifically an interface element, the PM window gives users a multiapplication environment in which to work. A user can have multiple applications open on one screen, allowing interaction, such as cut and paste, among many different types of applications. On the host, users can have at most two applications available at a time through the split screen capability.

First-time Presentation Manager programmers will find that the learning curve is long and difficult because of the large number and complexity of PM interface elements.

MESSAGE-BASED VS. STRUCTURED PROGRAMMING

In *Programming the OS/2 Presentation Manager*, Charles Petzold notes that the biggest hurdle in moving from a host to a PM interface is grasping the concept of messaging. While host interface programming is accomplished

through traditional structured programming, PM interface programming is done exclusively with messaging.

The main procedure of a PM application always contains the code fragment shown in Figure 1.

In the PM message loop, control flow in PM is done with message passing. Messages can be sent directly to a window or can be queued. Imagine an application where the interaction between the user and the window is accomplished by sending a message for each task that the user tries to do. Every key stroke or mouse click causes an invisible message to be sent to a message queue.

The message queue is an object in PM whose function is to store these messages and dispatch them to the appropriate window by switching control to associated window procedure. This loop continues to execute until the application is told to quit. A skeleton of a PM window procedure is shown in Figure 2.

The window procedure looks very much like a normal structured procedure. A message drops through and finds the appropriate case statement. If nothing is found, it falls through and control goes to a default window procedure. At the end of the message processing, a return statement returns control to the message processing loop.

A window can send messages directly to another window or send messages back to itself. This implies that a window procedure can call itself, requiring languages used within the window procedure to be reentrant.

Programming a host interface with DTL or an interactive system productivity facility is different. Panels are defined separately in panel definition files. Typically, control flow is more sequential and structured. For example, displaying a window is often part of a total **do-while** statement. Typical logic for displaying a panel is shown in Figure 3. An example of code that performs the task in Figure 3 is shown in Figure 4.

First-time Presentation Manager programmers will find that the transition to a message-based

architecture is a difficult one with a long learning curve.

OBJECT-ORIENTED VS. NON-OBJECT-ORIENTED

Presentation Manager is an object-oriented approach to creating interfaces, while the traditional host method of creating interfaces can be characterized as structured programming.



LOGIC FOR HOST DISPLAY OF PANEL

1. Initialize fields on panel.
2. Do while user does not hit exit:
 - a. Display the panel.
 - b. Check field one. If OK, continue. If not OK, send the user a message; go to top of loop.
 - c. Check field two. If OK, continue. If not OK, send the user a message; go to top of loop.
 - d. All fields are OK: Display second panel.
3. End do.

Figure 3: Logic for host display of panel

In PM, each interface element can be viewed as an object. For example, if a user moves the mouse pointer and clicks on the Exit button, the window disappears. In this situation, the user performs some action (clicking the mouse) on an object (the button), and a message is sent to the object receiving the action (the button). The receiving object then uses some method to interpret the action. In this case, the button interprets the action as a signal to make the window disappear.

PM can be characterized as event-driven programming and is actually a simulation of a sequence of events. In contrast, structured programming is characterized more by the traditional process-state method. A program is a set of instructions that saves values into storage locations (variables), continually modifies their values, and moves them to other storage locations. By examining these storage locations, the programmer determines the state of the program and the results.



```

main()
{

    /* 1. Initialize fields on panel. */

    BlankandNull(sizeof(Mfield1), Mfield1);
    BlankandNull(sizeof(Mfield2), Mfield2);

    /* 2. Do while user doesn't hit exit. */

    while (!ExitKey)
    {

        /* a. Display the panel. */

        rc = ISPLINK("DISPLAY ", "PANLMAIN", " ", reqfield);

        /* Get return code from panel display. */

        PanelRc = rc;

        /* If panel return code was 8, (exit or cancel), set ExitKey to */
        /* on. Otherwise, process main panel. */

        if (PanelRc == 8)
            ExitKey = ON;
        else
        {

            /* Convert value of ACTBRCH variable to integer so a switch */
            /* can be done on it. */

            SetValue(&ActionBarValue, ActionBarChoice);

            /* b. Check field one. If ok, continue. */
            /* Call routine to process field 1. */
            Field1Rc = Field1Process(&Request);
            if (!Field1Rc)
            {

                /* c. Check second field on panel. If ok, continue. */

                Field2Rc = Field2Process(&Request);
                if (!Field2Rc)
                {

                    /* d. Call routine to display second panel. */

                    ProcessNew(&Request);
                } /* end if
            } /* end if
        } /* end else
    } /* end while
    exit;
}
/* * End of main *

```

Figure 4: Sample host program for displaying a panel

In traditional programming methods, the programmer addresses such issues as complexity and reuse by using abstraction mechanisms such as procedures, modules, and abstract data types. Procedures allow programmers to execute different tasks repeatedly. Modules allow programmers to share data across an entire application and separate parts of an application to keep some information only accessible locally. Abstract data types, in turn, allow programmers to define their own data types similar to system-defined data types.

PM, as an object-oriented system, does not organize information in the same way. Data is thought of as objects and each object belongs to a group, called classes. Classes are then organized in a hierarchy where each class can be broken into subclasses. A class determines the behavior of an object, and subclasses are said to inherit the properties of their superclasses. Classes and inheritance allow reuse in PM. By organizing data in classes, code is reused by all instances of a class. Through inheritance, a subclass can automatically reuse methods that its superclass manifests.

Another difference between host and PM programming is how data is accessed. The only way an object can be accessed in PM is through the object's handle. A handle is a unique identifier for an object in PM. Typically, to perform an operation on an object, the programmer identifies the object through its handle and sends it a message. On the host, programmers instead associate variables with a particular interface element to store information about the element.

For example, to store text in an entry field and information about the entry field's maximum length, a programmer would typically create an abstract data type or store each in two variables. To get to this data, the programmer then has to access two storage locations. In PM, data is encapsulated. To access anything needed in an object, the programmer need only know its handle.

DYNAMIC VS. STATIC INTERFACES

A PM interface can dynamically change throughout the life of the application, while a host interface is static. The behavior of a PM interface element can change throughout the life of the program through messaging. An application can send a message to a control to change its state from write-mode to read-only. A PM window's appearance is always changing.

Presentation Manager allows developers to dynamically interrupt system processing to tailor controls or processing. On the host, panels are predefined and displayed in a fixed way.

In general, the difference in the way you define host interface elements and the way you define PM interface elements' behaviors is that PM interface elements' attributes and behavior are defined in the application program or predefined in a .DLG file. Host interface elements' attributes and behavior are always predefined in panel definition files.

Because you can change things dynamically with a PM interface, there is much more work involved in designing and coding it than for a host interface. As a PM programmer, you must spend more time designing the interface by determining which events will signal which changes to the interface. As a host programmer, you do not need to do this upfront design work or coding to implement the more complex design.

MULTIPROGRAMMING AND MULTITASKING

PM programming introduces the concepts of multitasking and multiprogramming. Host programmers will have to become familiar with various concepts to code a multitasking application. These concepts include semaphores, queues, threads, processes, and so on. These concepts are not trivial, but once learned can add lots of function and potential to an application.



The interaction between mouse and screen narrows the gap between user and program.



The biggest hurdle you will encounter is changing the way you think as an interface programmer.

A consideration when developing a hierarchical structured program vs. developing an object-oriented program is code review. It is easier to review code for a structured program. You simply follow the code's logic, stepping through each instruction in sequential order. For an object-oriented program, it is difficult to tell which object will be functioning at which time. For a PM program, each case in a switch statement may represent processing for an object. However, you cannot tell which case statement will get control at any given time.

It is difficult to debug a PM program when you don't know exactly when each statement gets control. Not only is the program processing case statements for new objects defined by the programmer, it is also handling PM system actions, such as creation of windows, system-generated messages, and so on. It is easier to debug a hierarchical structured program than an object-oriented one.

Porting a host workstation interface does not necessarily mean that data will also reside on the workstation. The programmer needs to think about where data will be residing and how much processing will be done on the host vs. the workstation. Design decisions must be made about what type of architecture will be used, whether it is client/server, cooperative processing, or distributed.

Performance, data integrity, and data recovery all become key issues when deciding whether to port a host application to the workstation. In fact, these issues are probably the biggest obstacles in deciding to port a host application to the workstation. If your application's interface and data repository both reside on the host, you can more easily ensure performance, data integrity, and data recovery. In deciding to move to the workstation, these issues become much more complex and difficult to implement.

Estimating effort and time management is difficult with PM programming. For the first-time PM programmer, there is a long learning curve. For the novice or experienced PM programmer, interface design involves a lot of research. There will be times where you will

not know how to do a specific type of implementation and research will be required.

There are lots of tricks in PM programming, and there are times when you think a task is difficult, but will actually turn out to be quite easy. There are times when implementation seems straightforward, but the actual coding turns out to be difficult. Since there is a long history of experience with host programming, its schedules are more easily estimated and adhered to.

CONCLUSION

There are many considerations when deciding whether to port a host application to the workstation. The decision to move to the workstation hinges on whether there are adequate tools available to create the workstation interface, provide the host-workstation communication protocol, and provide data integrity and recovery.

If you have already decided to port your host application to the workstation, the biggest hurdle you will encounter is changing the way you think as an interface programmer. However, learning to think like a PM programmer is becoming easier as PM programming tools proliferate. Currently, the WorkSet/2™ (the PM Developer's Toolkit) comes with a dialogue editor that automates panel design as well as generates code for your panel and an information presentation facility (IPF) that allows you to automate help-panel generation.

If you have not yet decided to make the transition to workstation interface, the decision must be made by weighing the advantages of a workstation interface against the cost of creating it. Even though the shift in thinking and the learning curve required to become a PM programmer make the cost of the transition high, the benefits gained by shifting to a PM interface are many. The PM interface is an intuitive, object-action, and event-driven interface that conforms to the criteria outlined in the SAA CUA91 Reference. The PM application model makes creating user-friendly, consistent interfaces easy. PM's object-orientation encourages principles such

as data abstraction and encapsulation, which contribute to the possibility of reuse and easier application maintenance.

Diane Lovelett, IBM Corp., Internal zip Y41E/921, Myers Corners Rd., Poughkeepsie, N.Y. 12603, (914) 296-6361. Lovelett is a senior associate programmer working on cooperative Presentation Manager applications. She joined IBM in 1988 as an information developer. She has a B.A. in English from S.U.N.Y. Buffalo and an M.S. in Technical Writing from Rensselaer Polytechnic Institute in Troy, N.Y.

Christine Grau, IBM Corp., Internal zip Y42/921, Myers Corners Rd., Poughkeepsie, N.Y. 12603. Grau is an associate programmer working on CASE solutions, focusing in cooperative processing Presentation Manager applications. She joined IBM as a summer hire in 1989 in the area of MVS development and joined full-time in 1990. She has a Bachelor's degree in computer science and economics from Rutgers University, N.J.

REFERENCES

Petzold, Charles. *Programming the OS/2 Presentation Manager*. Redmond, Wash: Microsoft Press, 1989.

OS/2 v. 2.0, volume 4: Application Development, (IBM Doc. GG24-3774).

SAA CUA91 Reference, (IBM Doc. SC34-4290).





GUI

Demystifying Custom Controls



Mark A. Bengé

by Mark A. Bengé and Matt Smith

Since the early days of OS/2 1.1, many developers have wondered how to create controls that act like the push button, scroll bar, and entry field controls provided by the system. Most of the necessary information has been present within the OS/2 reference manuals, but scattered throughout them. Only through deduction and painstaking observation of custom controls that operated exactly like those that the system provided could you create a custom control.

This article, the first of two parts, pulls together all the information found within the references and explains areas that have been missing. From this article and the accompanying example control source code, you will be able to create your own custom controls.



Matt Smith

WHAT IS A CONTROL?

Under OS/2 Presentation Manager (PM), a control is really just another window. In real terms, it is very similar to any client window you create in your current OS/2 PM applications. You need to register the class of a control just like you would with a client window, and the control is created in a similar manner to your client window through the `WinCreateWindow` function. The major difference is that the control can also be created through a dialogue template that resides in the application's resources.

CONTROLS IMPLEMENTATION PRIMER: THE MAGIC REVEALED

The makeup of a custom control is generally easy to follow and is dependent on the

complexity of the control. For example, a complex control like the container that is found within OS/2 PM contains approximately 42,000 lines of code, whereas our example image button contains almost 2,100 lines of code and most of those lines are comments.

The most critical issue facing a control is data encapsulation and reentrancy. This is easily solved since the designers of OS/2 PM allowed for window words that you reserve through the `WinRegisterClass` function shown in Figure 1.

The last parameter, `cbWindowData`, is used to specify the number of window words you need. For custom controls, you will need at least eight bytes, where the first four bytes are used as the `QWL_USER` user storage and the second four bytes for the internal control data that will record pertinent information about the instance of the control.

The internal control data structure is simply a structure that contains information about the control, such as size and position, current style, owner and parent window handles, and so on. You can place anything you deem necessary in this structure, since you totally control its contents, and it is private to the control.

This structure allows for data encapsulation and reentrancy. Figure 2 shows part of the control structure used for the image button and Figure 3 shows how you place the address of the structure within the reserved window words and how to retrieve it.

This is part of the magic, as we will call it, of creating a custom control that hasn't been fully

explained in the *OS/2 Programmer's Reference*. The second part of the magic is the messages that the control handles. Table 1 outlines most of the messages you may want to handle in your custom control.

The third part of the magic (and this is real magic since it isn't documented anywhere) is the presentation parameters. These magical mystical creatures provide some of the most powerful features found within OS/2 PM. The nice part about the presentation parameters is that when you are dealing with them in a custom control, you really don't have to concern yourself with them except where you want.

One area misunderstood with presentation parameters is how they arrive at the control. `WinCreateWindow` through the `pPresParams` parameter in Figure 4, allows for a set of presentation parameters to be passed to the control. A second method is through the resource script file as shown in Figure 5. Here, the values are preset, and they ultimately are passed to the control through the `WinCreateWindow` function when the control is created by the dialogue manager. The last method is through the `WinSetPresParam` function. In this case, the control won't receive a packet of presentation parameters like those through the `WinCreateWindow` function but will receive the `WM_PRESPARAMCHANGED` message.

Although the presentation parameters are passed to the control through the `WM_CREATE` message in the `CREATESTRUCT` parameter, they are set by OS/2 PM on behalf of the control and only those of interest need to be determined by the control. One method is to walk the presentation parameters list that is passed to the control through the `CREATESTRUCT` parameter and the `pPresParams` field. The only concern here is that the presentation parameter may have been set as an index value rather than a RGB color.

Therefore, you should query for the presentation parameters that are of interest to you and query for the RGB color values. This is done by using the `QPF_ID*COLORINDEX` flag with `WinQueryPresParams`. Therefore, when you do the painting of the control, you switch to RGB color mode from index mode using the function `GpiCreateLogColorTable` with the `LCOLF_RGB` flag, allowing you to use exact colors.

Even though the presentation parameters have been set for you by the `WinCreateWindow` function, you need to query for them since you will not receive the `WM_PRESPARAMCHANGED` message. You can either query for the presentation parameters of interest during the processing of the `WM_CREATE` message or each time you paint the control. We suggest that you do the former and record the values within the internal control data, making the painting of the final control as fast as possible.



```
WinRegisterClass ( HAB hab, PSZ pszClassName, PFNWP fnWndProc, ULONG
                  fStyle, ULONG cbWindowData );
```

Figure 1: `WinRegisterClass` definition

```
#define IBS_UP          0x0001    /* Button State: Up          */
#define IBS_DOWN        0x0002    /* Button State: Down        */
#define IBS_DISABLED    0x0004    /* Button State: Disabled    */
#define IBS_CAPTURE     0x1000    /* Button State: Capture     */

/* --- Internal Structures ----- */

typedef struct _IMGBTN          /* imgbtn */
{
    ULONG    id;                /* ID Value                  */
    ULONG    fStyle;            /* Style                     */
    BOOL     fFocus;            /* Focus Flag                */
    o
    o
    o
    HWND     hwndOwner;         /* Owner Window Handle      */
    HWND     hwndParent;        /* Parent Window Handle     */
} IMGBTN ;

typedef IMGBTN *PIMGBTN;
```

Figure 2: Internal control data structure

The `WM_PRESPARAMCHANGED` message will only be received when the application uses the `WinSetPresParam` function to add presentation parameters and `WinRemovePresParam` to delete the presentation parameter, or when the Scheme Palette for the application changes.

A good example of the message `WM_PRESPARAMCHANGED` can be demonstrated with the `PP_FONTNAME` presentation parameter. If



you use `WinSetPresParam(hWnd, PP_FONTNAME, 7L, (PVOID)"8.Helv")` to set the font of the control, you don't have to worry about loading the font and doing all the usual messy font handling. When you draw the text in the `WM_PAINT` message, the presentation space returned to you by the `WinBeginPaint` function will have already selected the font, so you only have to use `GpiCharString*` or `WinDrawText` to draw the text for the control. Since OS/2 2.0 now allows you to specify scalable fonts, you don't have to worry about those either.

```
#define QUCWP_WNDP (QWL_USER + 4) /* Pointer to Internal Control Data */

/* Allocate memory for internal control data */

DosAllocMem((PPVOID)&pingbtn, sizeof(IMGBTN),
    PAG_READ | PAG_WRITE | PAG_COMMIT);

/* Save the address of the internal control data */
/* in the control's reserved memory to allow it */
/* to be referenced as required by the control */

WinSetWindowPtr(hWnd, QUCWP_WNDP, (PVOID)pingbtn);
0
0
0

/* Get the address of the control info from the */
/* control's reserved memory */

pingbtn = (PIMGBTN)WinQueryWindowPtr(hWnd, QUCWP_WNDP);
```

Figure 3: Accessing reserved window words

```
WinCreateWindow ( HWND hwndParent, PSZ pszClass, PSZ
    pszName, ULONG flStyle, LONG x,
    LONG y, LONG cx, LONG cy, HWND
    hwndOwner, HWND hwndInsertBehind,
    ULONG id, PVOID pCtldata, PVOID
    pPresParams );
```

Figure 4: WinCreateWindow definition

HANDLING COLOR

Handling color is different. Only the foreground and background colors are automatically taken care of within the presentation space. When the foreground and background presentation parameters are set, the presentation space you receive from

`WinBeginPaint` will have those colors selected as the default colors. If the control you are creating only uses the foreground and background colors, you do not need to record the colors within your internal control data. If you want to use the disabled and highlight values, you need to query for them. That can be done during processing of `WM_CREATE` or just before you paint the control.

When the application uses the `WinSetPresParam` or `WinRemovePresParam` functions, the control will receive the message `WM_PRESPARAMCHANGED` with an index value of the presentation parameter that has changed. Now, you can determine which presentation parameter index or RGB color value has been changed and whether or not you should query for the color value using the `WinQueryPresParam` function with the passed presentation parameter index value.

You can query the color added or removed based on index value (that is, `CLR_*` or `SYSCLR_*`) or as a RGB color value using the `QPF_ID*COLORINDEX` flag. We suggest that you use the RGB color value since it will allow you to better control and display colors within the control. The color will be returned when a presentation parameter is present, and `WinQueryPresParam` will return the number of bytes returned. When a presentation parameter has been removed through the function `WinRemovePresParam`, `WinQueryPresParam` will return 0L, indicating no presentation parameters are present for the parameter requested. In this situation, the control should revert back to its default color scheme for that parameter.

Special consideration should be given to a condition that is not properly documented: the indication of a global presentation parameter update that happens when the user applies a new color scheme to the window or dialogue box using the Scheme Palette. The message `WM_PRESPARAMCHANGED` will be received by the control with a presentation parameter index value of 0 (contained in `mp1`). When received, the control should query the presentation parameters of interest to make sure that any color changes are recorded.

When placing presentation parameters within resource script files, the `PRESPARAMS` statement is used. It allows the presentation parameters for the control to be specified so that when the



COMMAND	MESSAGE USAGE
BM_CLICK	Simulates a button click. It is used in conjunction with WM_QUERYDLGCODE and WM_MATCHMNEMONIC .
WM_ACTIVATE	Registers or deregisters with the Help manager.
WM_ADJUSTWINDOWPOS	Informs the control that its position or size is about to change. It allows the control to adjust the values if required.
WM_*SELECT	Informs the control of the starting and ending of swipe selections.
WM_BUTTON*DOWN	Used to monitor mouse button presses.
WM_BUTTON*UP	Used to monitor mouse button releases.
WM_CHAR	Used to monitor keyboard selections for the control.
WM_CREATE	Performs control initialization, allocates internal data structures, performs initial sizing of control, presentation parameters, and control setup.
WM_DESTROY	Provides control clean-up such as releasing resource and memory objects used.
WM_ENABLE	Monitors when the control is becoming enable or disabled.
WM_ERASEBACKGROUND	Used to erase the control background.
WM_HITTEST	Used to respond to mouse handling of the control.
WM_MOUSEMOVE	Used to monitor mouse pointer movement.
WM_PAINT	Used to paint the control.
WM_MATCHMNEMONIC	Used to determine if a typed character matches a mnemonic in its window text.
WM_PRESPARAMCHANGED	Used to monitor changes in presentation parameters that are applied to the control by the system.
WM_QUERYCONVERTPOS	Used to provide for conversion of DBCS characters.
WM_QUERYDLGCODE	Used to identify the capabilities of the control to the system.
WM_QUERYWINDOWPARAMS	Used to process window parameter queries such as length of text and actual text retrieval.
WM_SETFOCUS	Used to monitor when the control is receiving focus or losing it.
WM_SETSELECTION	Used to highlight or dehighlight a selected item.
WM_SETWINDOWPARAMS	Used to process window parameter settings such as setting text.
WM_SIZE	Used to monitor size and position changes being performed on the control.
WM_SYSCOLORCHANGE	Used to inform the control that a system color has changed.
WM_WINDOWPOSCHANGED	Used to inform the control that its position has changed.

Table 1: Messages relevant to custom controls (messages in bold are used in image button example)



```
CONTROL " ", EF_TEXT, 262, 25, 46, 8, WC_ENTRYFIELD,
                ES_MARGIN | WS_TABSTOP | WS_VISIBLE

CTLDATA 8, 6, 0, 0
PRESPARAMS PP_FOREGROUNDCOLORINDEX, CLR_RED
PRESPARAMS PP_FONTNAMESIZE, "8.Helv Italic"
```

Figure 5: Resource script showing PRESPARAMS

```
POINTL    aptl[2];          /* Dialog Units Conversion Points */
ENTRYFDATA efd;             /* Entry Field CTLDATA */
PPRESPARAMS ppres;          /* Presentation Parameters Array */
PPARAM    pparam;           /* Parameters Pointer */

efd.cb = sizeof(ENTRYFDATA);
efd.cchEditLimit = 6;
efd.ichMinSel = 0;
efd.ichMaxSel = 0;

aptl[0].x = 262L;
aptl[0].y = 25L;
aptl[1].x = 46L;
aptl[1].y = 8L;

WinMapDlgPoints(hwndParent, aptl, 2L, TRUE);

DosAllocMem((PPVOID)&ppres,
            sizeof(PRESPARAMS) + sizeof(PARAMS) * 2 + 16,
            PAG_READ | PAG_WRITE | PAG_COMMIT);
ppres->cb = sizeof(PARAMS) * 2 + 16;
pparam = ppres->aparam;
pparam->id = PP_FOREGROUNDCOLORINDEX;
pparam->cb = 4L;
pparam->ab = (BYTE)CLR_RED;
pparam = (PPARAM)(ppres->aparam + (BYTE)(sizeof(PARAM) + 3L));
pparam->id = PP_FONTNAMESIZE;
pparam->cb = 14L;
memcpy(pparam->ab, "8.Helv Italic", 14);

WinCreateWindow(hwndParent, WC_ENTRYFIELD, "", ES_MARGIN | WS_TABSTOP | WS_VISIBLE,
                aptl[0].x, aptl[0].y, aptl[1].x, aptl[1].y, hwndOwner,
                HwndBottom, EF_TEXT, (PVOID)&efd, (PVOID)ppres);
```

Figure 6: WinCreateWindow with presentation parameters example

control is created through a resource script in a dialogue box or window, the desired presentation parameters are used. Figure 5 shows the method of doing this with an entry field where the font desired is 8-point Helv Italic with the foreground color red.

Presentation parameters must be in the proper format to allow them to be properly set, as shown in Figure 6. Also, presentation parameters are inherited from the parent. In this case, the control should query the presentation parameters to see what they are

by using the `WinQueryPresParam` function without the `QPF_NOINHERIT` flag.

The next magical area to consider is the keyboard. How does the control know which control to move to when the user presses the tab key? Well, by handling the `WM_CHAR` message and looking for the `VK_TAB` key press, you only need to use the `WinEnumDlgItem` function to determine the next tab item. Likewise, when the user presses a cursor key, he or she uses the same function to determine the next or previous group item.

SETTING UP THE CONTROL

There are two ways to set up the control: during initialization and with messages. You will see throughout the OS/2 PM architecture that both methods are implemented. For example, you can use the `ENTRYFDATA` structure to set up an entry field for the maximum length of text the control can handle along with the starting and ending positions of selected text. This structure is only used during the entry field's creation. If the structure is not present, the control is still created but uses certain defaults. The same task of setting the maximum input text and selection positions can be accomplished by sending messages to the control. For the setting of the text length, you would use the `EM_SETTEXTLIMIT` message, and, for the text selection, you would use the `EM_SETSEL` message.

Good control design implies that the control should be able to handle a `CTLDATA` type structure, of which the `ENTRYFDATA` structure is a subset, whereas many of the static setup functions that a control needs to perform are defined within such a structure. For example, a control within OS/2 PM that does not follow this is the spin button, where you need to specify to the each of the servant controls who the master is through a message as shown in Figure 7. This could have been easily done through a `CTLDATA` type structure like the one shown in Figure 8, where, if the style of the spin button is a servant, the master owner of the control is specified in the structure and used when the control is created. Figure 9 shows this method in a resource script file.

The common element here is the ID of the master spin button. During the `WM_Create`

handling of the spin button, the window handle of the master could have been determined exactly in the same manner that Figure 7 does now with just a little further work.

The basic goal is to minimize the amount of work you have to do to set up the control within your final application. This is particularly relevant when you use the control within dialogues, and the information is recorded within a dialogue template.



```
WinSendDlgItemMsg(hWnd, SPB_SERVANT, SPBM_SETMASTER,
    MPFROMHWNID(WinWindowFromID(hWnd, SPB_MASTER), 0L);
```

Figure 7: Example message

```
typedef struct _SPBCDATA {
    USHORT cb;           /* Structure Length */
    USHORT idMaster;     /* Master Spin Button ID */
} SPBCDATA ;
```

Figure 8: Possible spin button `CTLDATA` structure

```
CONTROL " ",          SPB_SERVANT, 40, 182, 15, 12,
                      WC_SPINBUTTON, SPBS_SERVANT |
                      SPBS_JUSTCENTER | SPBS_NUMERICONLY |
                      SPBS_PADWITHZEROS | WS_GROUP |
                      WS_TABSTOP | WS_VISIBLE
                      CTLDATA 4, SPB_MASTER
```

Figure 9: Possible spin button resource script file for `CTLDATA` implementation

EXAMPLE IMAGE BUTTON IMPLEMENTATION

Figure 10 shows a custom control we will create as an example. This control is similar to the push button except that it allows for a bitmap image and text as integral components of the control. The image button control allows for two styles: `IS_TEXTONBTN` for Button 1 and `IS_TEXTBELOWBTN` for Button 2.

Basically, we wanted the image button to allow the user to interact with it either through the keyboard or mouse. It had to allow for the images to be specified at the time of creation and at a later point through a message. As for



the images, three distinct images had to be allowed: button up, button down, and disabled.

By understanding these basics, we can now begin the implementation of the image button. Since we can't show all of the source code here, we will highlight the areas we think are important. We suggest that you try to gain access to the source code, which can

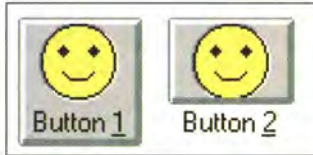


Figure 10: Example custom control

A design goal for the button was that when the button was in the down position, a specific image could be shown if required. Also when the control was disabled, a third distinct image could be shown. Figure 11 shows the implemented controls within the test applet dialogue.

In Figure 11, Button 1 is shown in its normal up position. The button is enabled and can be selected either with the mouse or the keyboard. Button 2 is shown in its disabled state, Button 3 shows emphasis, and Button 4 shows the button when it is depressed.



Figure 11: Example dialogue containing custom controls

The CTLDATA structure defined for the image button is shown in Figure 12 and is very simple in design. The only rule, which is not sufficiently documented, regarding the CTLDATA structure is that the first field of the structure must be the size, which, in the case of the image button structure, is the `cb` field. OS/2 PM uses this value to allocate memory for a copy of the data that is passed to the control in the `WM_CREATE` message.

The field should not contain a value larger than 65,535, as the system will only allocate a 64KB memory object. Since this limit may be lifted in future versions of OS/2, you may wish to set up the field as a `ULONG`, but limit the data structure size to be less than 64KB as we have done. This will allow us to expand the structure in the future to include new functionality. The corresponding resource script for the CTLDATA statement is also shown in Figure 13, and Figure 14 shows the equivalent source implementation.

While we are all quite familiar with this outside view of the control, it is the inside part of the control that is the mysterious part and what we will now dive into.

Part 1 of the control window procedure source code handles the `WM_CREATE` message, which is the first message it receives when it is created. Here is where we need to perform the control initialization and get the control ready for display. Since the source code is too long to show here, we will briefly describe what actions we perform in the creation sequence. First, we check to see that the CTLDATA structure that may be passed in `mp1` is valid. This is done by checking the structure length element

```
/* --- CTLDATA Structure Definition ----- */
/*****
/* NOTE: IDs for the bitmaps must be 0 < id < 65535
/* This limitation is due to OS/2 PM Version 2.0
*****/

typedef struct _IMAGEBUTTON /* ibtn */
{
    /* Size: 16 Bytes */
    LONG cb; /* Structure Size */
    LONG idBitmap; /* Bitmap ID : Normal or Up Position */
    LONG idBitmapDown; /* Bitmap ID : Down Position */
    LONG idBitmapDisabled; /* Bitmap ID : Disabled */
} IMAGEBUTTON ;
```

Figure 12: Image button CTLDATA definition

be found on CompuServe in LIB13 of the OS2DF2 forum or through the other electronic means outlined at the end of the article, since it fully implements the control and also provides a little test applet to exercise the control.

cb to make sure that it is the correct value. If this value is correct or if the **CTLDATA** structure was not used, we allocate the internal control data memory that we need to be able to operate the control.

The address of the structure is then placed within the reserved window words of the control. This will allow us to retrieve the data for the control as we require it when we process any messages of interest.

When the **IMAGEBUTTON CTLDATA** information is present, we begin loading the bitmap images specified within structure. We make the assumption that the images are part of the executable resources and try to load the required bitmap image from those resources and create the bitmap handle for the image. Each bitmap image handle is saved within our internal control data so we can properly paint the required image on the button when required.

If the bitmap images are not present or specified, we load the default images from the dynamic link library (DLL) that contains the implementation of the image button control, but you don't have to implement a custom control in a DLL, you can implement it within an application. Most likely, you will want to make the custom control you create available to all the applications you create.

After we process **CTLDATA**, we process the information contained within **CREATESTRUCT** that is passed to the control in **mp2**. This structure contains information as to the size and position of the control, the owner and parent of the control, the window style, text, and presentation parameters. You will need to process this information, especially the size and position, since you will not receive a **WM_SIZE** message as part of the window creation sequence. Also, save the owner and parent window information if you want to communicate back to either of these windows in the form of notification messages. In our case, we save this information within our internal control data.

At this point, we also determine if presentation parameters are present by querying for those of interest, since we will use the colors defined later in painting the control.

Finally, we perform the sizing operations on the control, calculate the various coordinates of the elements. These items are the polyline arrays, the text, and bitmap locations.

Part 2 of the control source code handles control text and colors. Text setting and retrieval are handled through the **WM_SETWINDOWPARAMS** and **WM_QUERYWINDOWPARAMS** messages. The **WM_SETWINDOWPARAMS** message



```
CONTROL "Button 1",      IB_BUTTON1, 15, 25, 35, 27, "ImageBtn",
                        IS_TEXTONBTN | WS_TABSTOP | WS_VISIBLE
                        CTLDATA 16, 0, 100, 0, 101, 0, 102, 0
```

Figure 13: Image button **CTLDATA** definition and resource implementation

```
POINTL    aptl[2];      /* Dialog Units Conversion Points */
IMAGEBUTTON ibtn;      /* Image Button CTLDATA */

ibtn.cb = sizeof(IMAGEBUTTON);
ibtn.idBitmap = 100L;
ibtn.idBitmapDown = 101L;
ibtn.idBitmapDisabled = 102L;

aptl[0].x = 15L;
aptl[0].y = 25L;
aptl[1].x = 35L;
aptl[1].y = 27L;

WinMapDlgPoints(hwndParent, aptl, 2L, TRUE);

WinCreateWindow(hwndParent, "ImageBtn", "Button 1",
                IS_TEXTONBTN | WS_TABSTOP | WS_VISIBLE,
                aptl[0].x, aptl[0].y, aptl[1].x, aptl[1].y, hwndOwner,
                HWND_BOTTOM, IB_BUTTON1, (PVOID)&ibtn, (PVOID)NULL);
```

Figure 14: Source code implementation

takes any new text for the control and saves it within the internal control data; the next time the window is painted, the new text will be displayed. Conversely, **WM_QUERYWINDOWPARAMS** returns either the length of the current text for the control or the actual text, depending on the request.

When the control is sized through the application explicitly, Part 3 of the code

Ad Index

American Computer Technology
Arcadia
Bon Ami
Borland International Inc.
Borland International Inc.
Burton Systems Software
Caseworks
Computer Language
Computer Security Institute
Creative Systems Programming
D Soft Development
Database Programming & Design
DeScribe
Essex
Gpf Systems
Hamilton Laboratories
IBM Canada
IBM Tools
IBM OS/2 Division
IBM LAN
IBM Controls Library

55	IBM IV League	73
19	IBM OS/2 Conference	95
55	IBM OS/2 Developer	61
Cover 4	Intelligent Environments	21
33, 32a-b	Micro Focus	2
72	Micro Way	72
7	Mitnor	67
42	One-Up	79
62	Princeton Windowing Systems	55
74	Quadron	51
128	Quercus	128
68	SE International	51
35	Soft & GUI	51
52	Softbridge	15
77	Software Development Conference	47, 93
74	Sourceline Software	74
13	The Stirling Group	Cover 3
16a-b	Sundial Systems Corporation	37
28-29, 49	XVT Software	30
38	Watcom	Cover 2-1
59		

dbfLIB

Programmer's Library



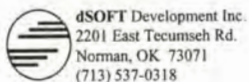
Do you need uniform access to dBASE files from your custom DOS, Windows, and OS/2 applications?

Introducing **dbfLIB**, the dBASE file access library for DOS, Windows, and OS/2 programmers.

Features:

- * A single set of functions for DOS, Windows, OS/2 1.3, OS/2 2.0
- * Royalty free Dynamic Link Libraries for:
 - Windows 3.0 and Windows 3.1
 - OS/2 1.3 (16-bit DLL)
 - OS/2 2.0 (32-bit DLL)
- * Static libraries for DOS and OS/2 applications.
- * Meaningful function names familiar to the xBASE programmer.
- * File and record locking.
- * Handle-based calls allows multiple instances of the same dbf file (very useful in multi-threaded OS/2 programs).
- * Create, update, and index dBASE files.

dbfLIB includes Windows DLL, OS/2 1.3 DLL, OS/2 2.0 DLL, and static libraries for DOS and OS/2.



REXX is the key that unlocks the full potential of OS/2

PERSONAL REXX™ is the complete REXX package for OS/2

If you're just getting into REXX, **Personal REXX** offers:

- A full 32-bit implementation of REXX that is the fastest available
- Complete documentation in the package
- Free support by telephone, BBS, and CompuServe
- Seven years of experience with REXX on a desktop platform

But if you are already a REXX expert, there's much more:

- True system-wide global variables
- Interprocess communication support for named pipes, semaphores, and the REXX external data queue
- Array operations like copying, sorting, and searching
- Advanced mathematical functions (trigonometric, logarithmic, exponential)
- Text-mode user interface support for windows, menus, and data entry screens.
- Date format conversion, regular expression file searching, and many other system services
- Full compatibility with IBM's OS/2 and mainframe REXX implementations

If you prefer, most functional enhancements are also available separately in our **REXXLIB™** add-on function package for OS/2 REXX.

Personal REXX is also available for DOS and Windows.

QUERCUS SYSTEMS

P.O. Box 2157
Saratoga, CA 95070
Phone: 408-867-REXX
Fax: 408-867-7489
BBS: 408-867-7488

handles the `WM_SIZE` message that is used to determine the new size of the control and recalculate the coordinates of the various drawing elements.

Part 4 of the source code is used to handle the enabling and focus of the control. The `WM_ENABLE` message is processed to allow the control to know when it has been enabled or disabled and how it should paint itself due to this. By recording the state of the control, the paint process can select the appropriate bitmap to display.

Likewise, the `WM_SETFOCUS` message is processed to allow the control to know when it has received or lost focus. This too allows the control to record the state so that when the paint process draws the text, emphasis can be removed or added to allow the user to know that the control has focus or not.

User interaction with the control through the mouse and keyboard is contained in Part 5 of the source code. When the mouse button is pressed, we process the `WM_BUTTON1DOWN` message. In this processing, we denote in the internal control data that the button is now in the down state and that we are also in capture mode. Why Capture mode? Capture mode allows us to determine when Button 1 of the mouse has been released. But isn't the `WM_BUTTON1UP` message sent to the control when the button is released? Well, yes and no. Confused?

Mouse processing is really quite simple; it works just like a normal application window. Your window receives mouse messages sent to your window *only* when the mouse pointer is on top of it. But we want to know when the button is released, *no matter where* the pointer is. When the mouse is within the limits of the control, we will receive the button down and up messages. But if the user presses Button 1 on the mouse within the limits of the control, keeps the button down and moves the mouse pointer outside the limits of the control, and releases it, the button-up message will be sent to the window in which the mouse pointer is now over, and the control will miss the button-up event. Therefore, by putting the mouse into mouse capture mode (`WinSetCapture`), when Button 1 of the mouse is pressed and only releasing the capture after Button 1 of the mouse has been released, we are assured of

knowing the button-up event has occurred.

When the button is released, we send a `WM_COMMAND` message to the owner window of the control to denote that the button has been selected. This is similar to the manner in which push buttons operate in OS/2 PM.

For the keyboard, the `WM_CHAR` message is processed. Here, we check for the tab and cursor movement keys. Depending on the keystroke, we use `WinEnumDlgItem` for the next or previous control in which the focus should be placed on, since the user is indicating that he or she wishes to move to a particular control.



MESSAGE	PURPOSE
<code>IM_LOADBITMAPS</code>	Loads requested bitmaps.
<code>IM_RESETBTN</code>	Resets the button to the up position.
<code>IM_SELECTBTN</code>	Selects the button and sets it in the down position.
<code>IM_SETBITMAP</code>	Sets a specific bitmap image using a bitmap preloaded by the application.
<code>IM_BTNCLK</code>	Simulates the click of the mouse pointer on the button, whether in the up or down position. Changes the state from up to down or down to up.

Table 2: Image button messages

It is also in this part of the code that we process the mnemonic selections for the control. When the message is received, we check to see if the control contains a mnemonic by looking for a tilde (~) in the text of the control and if found, check the next character against the character provided by the message. When a match occurs, we return `TRUE` which causes the system to send the `BM_CLICK` message to the control. This occurs because we responded with `DLGC_PUSHBUTTON` to the `WM_QUERYDLGCODE` message. When we receive the `BM_CLICK` message, we process the message on the basis that the button has been pressed, sending the `WM_COMMAND` message to the owner of the control.

The painting of the control is handled through the `WM_PAINT` message, which is Part 6 of the source code. This is an interesting message to handle since the OS/2 references publications do not document what a control should do.



The simple rule of thumb is that the background erasure is handled on your behalf by the owning window or dialogue, and the drawing into the control area uses the presentation space of the owning window. Hence, any attributes present in the owning window will be inherited. Also, you don't need to worry about fonts and font sizes, since these can be taken care of through the `PP_FONTNAME_SIZE` presentation parameter. The logical font and size will be automatically set for the presentation space, including scalable fonts.

For the most part, the painting process is just the process of drawing the control. In the case of the image button, it is drawing the bitmap image that is to appear on the button. This image is determined in the drawing routine, since the state may have changed since the last drawing. In this case, it may mean selecting the bitmap for the up position, down position, or disabled state.

Once the bitmap image is drawn using the information that was processed during the sizing operations, the beveled edges of the control are drawn. Using the coordinates calculated during the sizing operations, a very fast drawing mechanism is implemented here.



First, we set the drawing color and line types using `GpiSetColor` and `GpiSetLineType`. Next, we set the initial starting coordinate (in this case in the lower left corner of the bottom window) with `GpiMove`. Then we draw as many lines as necessary with `GpiPolyLine`. We draw the beveled edges of the control until they are complete and then draw the button border using a similar method.

Last, we draw the text of the control and, if emphasis is required, we draw the emphasis dots around the text using the `GpiSetLineType`, `GpiMove`, and `GpiPolyLine` functions. You can, during the paint processing, check to see if the style of the control has changed, in which case you may want to recalculate the various component coordinates before painting and save the new style internally. In essence, what you are trying to do here is use a cache of information to allow the painting to be as fast as possible without having to recalculate the components of the control each time.

Part 7 of the source code contains the message handling for the control specific messages sent from the application to the control. These messages, shown in Table 2, are designed to allow the application that owns the control a means of setting the appearance of the image button. The `IM_LOADBITMAPS` message, which is used to load bitmaps from the executable file, uses the same method within the `WM_CREATE` message. It allows you to dynamically change the images for an image button without having to destroy and recreate it. The messages you send the control can be anything you require and should be extendable.

The last part of the source code, Part 8, contains the `WM_DESTROY` message. When we receive this message, we know that the control is being destroyed, and we should delete the loaded bitmaps and release the memory allocated.

CONCLUSION

Knowing that controls are in fact just windows and that certain operations are performed on their behalf, it is easy to implement custom controls for applications. For example, you can easily create a pattern custom control that is based on the `GpiSetPattern` function or a simple line control based on the `GpiLine` function. Creating controls is not all that difficult if you follow a few simple rules. Using the example control that is the basis of this article, you can create your own control in a very short period of time. In our next article, we explain in more detail those mysterious presentation parameters and the design of a custom control.



REFERENCES

OS/2 2.0 Programming Guide Vol. II, (S10G-6494)

OS/2 2.0 Presentation Manager Programming Ref. Vol. I, (S10G-6264)

OS/2 2.0 Presentation Manager Programming Ref. Vol. II, (S10G-6265)

OS/2 2.0 Presentation Manager Programming Reference Vol. III, (S10G-6272)

The complete source code for the sample program in this article can be downloaded from the following electronic sources:

- In Europe, the source code is available on the International OS/2 Users Group bulletin board service at (44) 0454 633 197 or (44) 0454 633 420.
- In the U.S., call the IBM National Support Center bulletin board service at (404) 835-6600. The source code is in file area 11, named OS2DCTRL.ZIP.
- In Canada, call the IBM bulletin board service at (416) 946-4255 (Toronto), (514) 938-3022 (Montreal), or (604) 664-6466 (Vancouver). The source code is in file area 30 in the file OS2DCTRL.ZIP.
- On CompuServe, the source code is in LIB13 of the OS2DF2 forum.
- On VM (IBM node) source code can be accessed by issuing the following command:
REQUEST CUSTCTRL FROM BANZAI AT CARVM3.

Mark A. Benge, IBM Programming System Laboratory, 11000 Regency Pkwy., Cary, N.C. 27512-9968. Benge is a senior associate programmer who joined IBM in 1989. He received a B.S. degree in computer science from Western Carolina University. He works in CUA Controls development, where he has converted various controls to Microsoft Windows and converted 16-bit tools for OS/2 2.0 to the 32-bit environment.

Matt Smith, Prominare Inc., 100 Front St. E., Toronto, Ont., Canada M5A 1E1. Smith is lead architect for the Prominare Development System, an OS/2 2.0 advanced GUI development environment. He has been actively involved with OS/2 since 1988, cofounding Prominare in 1990. He has a degree in architecture from the University of Waterloo.



Database Manager

Developing Applications With Query Manager



Alan Eisen

by Alan S. Eisen

The OS/2 Query Manager component of Extended Services for OS/2 has many facilities for creating sophisticated database applications. Through the use of panels, menus, and procedures, many applications can be developed without additional tools. This article is intended to provide the beginning developer with some tips for using these facilities and linking them together to create complete applications. These tips were gathered over the last two years as I developed an application to manage two internal IBM programs, Systems Assurance and Critical Situations (Critsit).

USING PANELS

Panel Design

Panels are used to ease data access. They allow developers to label and position fields on the screen to make them meaningful to users.

The amount of data on any one screen should be limited to increase readability, but panels can be continued well beyond a single screen. Maximizing the Query Manager session when designing panels and working with a full screen rather than an arbitrarily sized window gives the user a reference point from which to see the entire panel. The user can run the application in a window, but if it is designed for a standard-size screen the user will know the maximum boundary.

When establishing an initial screen design, the easiest method is to select **Specify/Default Definition**, which tells the Query Manager to build the default panel with each field listed along the left side of the screen. This eliminates the need to mark each field and its text, as they are already on the screen and can be

individually manipulated. This is most convenient when large numbers of fields are involved, as each field and text item must be marked individually. The disadvantage is apparent when fields must be moved from one screen to another. Once the default is established you can continue the panel design, specifying subtables and lookup tables and renaming the fields.

As you position fields on the screen, it is helpful to align information vertically and horizontally to make it more readable. For example, a user can add information for a systems assurance review, as shown in Figure 1. Information about each review type is aligned horizontally. The dates, risk assessments, and waivers for all reviews are aligned vertically. This matrix of information makes completion easier for the user and allows fields to be described in groups rather than individually.

Linking Panels

The Critsit application, shown in Figure 1, requires a lookup table that contains IBM product information. It is more efficient to store descriptions of each product once than to repeat the descriptions for each row in the primary table. On occasion, the Critsit table will need to reference a product that has not yet been entered into the product table. To a new developer, it may appear that there is no mechanism for linking one panel to another. For example, when you press PF4 in the Panel/Operation Command column, there is no option to run another panel.

The way to do this is to use the **RUN PANEL** or **RUN PROC** command from the Panel/Operation Command column, which allows you to dynamically link from one panel to another. It is

Query Manager lets you develop applications without additional tools.



then possible to jump out of the Critsit panel into the product panel, enter the new product, and return just where you left off without losing data.

Function Key Standards

As you create panel operations, it helps to standardize function keys for the advanced user. A convenient group of function keys to use is Shift + PF4/5/6/7, which are unused keys for panels that can be assigned to your application. Consistency is important with these keys. For example, PF4 should always take you to the product table, whether it is accessed from the Critsit panel or the Systems Assurance panel. The Query Manager provides a good example of moving between forms, reports, and queries.

Initializing Panel Value Fields

After the individual panels are designed, it may be helpful to initialize the values of select fields (for example, by setting a timestamp field with the current timestamp). Creating a procedure to be run as the panel's Instance Rule (Add) allows you to initialize panel field values. Figure 2 shows a simple example that establishes a variable, *temp*, with the current timestamp. The DATETIME panel field is set equal to *temp*. Because the instance rule is run before each new panel is displayed, the value is always updated. If the field is not going to be altered by the user, it doesn't need to appear on the panel, although it must be defined in the table fields option.

If your timestamp column requires a unique value and there are simultaneous system users, this method will not guarantee that another user will not have the same value. If two users have the same value, the Database Manager will not allow the second row insert and the user will have to exit the panel and begin again.

As you become comfortable with linking panels and creating system variables, it will become obvious that some information from the first panel belongs in succeeding panels. You can capture this information for the user, avoiding data reentry. A simple example of this situation shows up whenever you attempt to add new products associated with a specific critical situation. The user may choose to add a new product to an existing Critsit; the PULL_VAR_CRIT procedure is executed instead of

the RUN_PANEL action in the panel operation command. It captures the value of the current KEY value (the Critsit number), stores it in a global variable, and runs the panel to add the new product. The second panel, CRITSIT_PROD, has an instance rule (Add) that runs the PUSH_VAR procedure, which fills the KEY value with the value taken from the Critsit panel. This simple technique, which combines panels and procedures, allows the user to move easily between functions without losing or reentering data. The PULL_VAR_CRIT and PUSH_VAR procedures are shown in Figure 3.

Figure 1: Display of screen layout

```
** This procedure captures the current timestamp in the
** variable temp and then moves the temp value into the
** panel field DATETIME. It is used as part of a Panel
** Instance (Add)*/
```

```
'GET CURRENT (temp=TIMESTAMP) '
'SET CURRENT (DATETIME = temp) '
```

Figure 2: Initializing a field

USING MENUS

Menus are created much like panels, except you define menu actions rather than fields. Menu actions can be linked directly to panels, queries, procedures, or other menus. You may also run other valid Query Manager commands as listed in the Help panel. Like



panels, menus should be carefully designed so their functions are obvious. Figure 4, an example of what not to do, shows a menu that tried to squeeze too much into one screen.

```

Procedure: PULL_VAR_CRIT
/*This procedure takes the KEY value from the starting panel and
**stores it into a global variable and then calls the next panel
**for data entry*/
'GET CURRENT (var1 = KEY)'
'SET GLOBAL (glvar = var1)'
'RUN PANEL CRITSIT_PROD (MODE=ADD)'

Procedure: PUSH_VAR
/*This procedure takes the global variable "glvar" and moves the
/*value into the current variable KEY. It is used in the Panel
/*Instance (Add) to pre-fill the field.*/
'GET GLOBAL (temp=glvar)'
'SET CURRENT (KEY=temp)'

```

Figure 3: Moving a value between panels

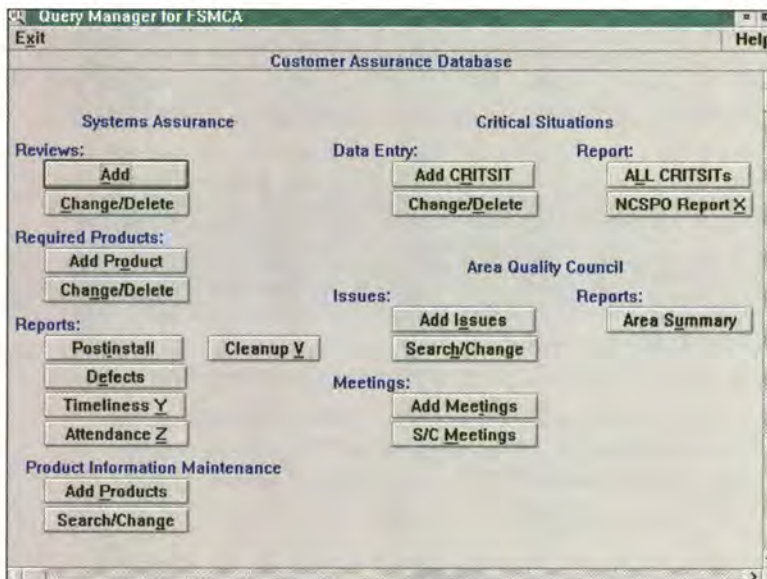


Figure 4: Menus—what not to do

A better design for the menu in Figure 4 would be to take the three major components, Systems Assurance, Critsit, and Area Quality Council, and make each one a menu selection. The menu action for these three items links to three separate menus with more room for options. Each menu item must also have a unique single-character mnemonic. Unfortunately, the short names that were used did not provide enough unique values so random letters such as Y and Z were used. This does not make for either an

attractive screen or mnemonics that are meaningful to the user.

A final benefit of nesting menus is the ability to control user access. In Figure 4, all functions of the database are displayed. A user who only needs access to Critsit still views and can still choose the other selections. Although database manager security (GRANT/REVOKE) prevents access, users tempted by the function will receive only a marginally helpful error message from the system. A nested menu allows users with access to the whole system to see all selections, while those with limited access can be started at the submenu.

LINKING WITH PROCEDURES

Query Manager procedures are not a substitute for a high-level language, as you cannot directly incorporate SQL statements within the procedure. They do, however, provide program control between other Query Manager functions such as queries and forms.

As noted in the documentation, a procedure must begin with a comment, delineated with the characters `/*` and `*/`, as in a REXX or PL/I program. Proper documentation within the program makes it easier for someone else to understand what you have done or for you to understand it when you come back in six months. These comments are not the same as comments in an SQL query, which precede the comment with two dashes.

One use of procedures is demonstrated by the Systems Assurance application, which creates separate reports for multiple IBM marketing branches. Because the reports need to be distributed electronically, a separate file for each branch is needed. A query that sorts by branch is not the answer, as it creates only one file.

Figure 5 shows a simple procedure that sets the value of a variable, `BRANCH`, for each run of a single query. The form used to print the output uses the same variable to print the branch name on the report. The `PRINT REPORT` lines are shown as continuation lines for printing purposes only. Continuation lines are not supported.

The procedure and query editors support OS/2 Cut and Paste facilities. To copy lines in the

procedure, highlight the selected text and press Ctrl+Insert. To paste the text, use Shift+Insert.

Multiple Queries vs. Row Logic

While using a REXX program to perform individual row manipulation is effective, it may be more complicated than necessary. Procedures and queries do not support host variables, which may make row manipulation difficult. However, if you look at your program differently and create a query that selects a unique set of rows, procedures and queries may be used exclusively.

Perhaps you want to import new data into a temporary table before either inserting it into an existing table or update existing rows. The REXX methodology would read each row from the temporary table, see if a corresponding row exists in the target table, and then perform an insert or update. A simpler method that can be written solely in Query Manager is to write a query that does an update where the two tables intersect and another query that inserts rows when they don't. A procedure can be written that first runs the import and then the queries and that can be run as needed.

This allows for the definition of a few queries that select just the data that satisfies each required case and manipulates all the rows. Rethinking processes to satisfy single or multiple queries allows the Database Manager to perform more of the work, which makes the application more efficient and easier to develop.

CREATING SEQUENTIAL NUMBERS

When there is no obvious way to create a unique row value, the easiest way is to create a sequential number. While the Query Manager will not create sequential numbers automatically, it is easy to do. The following method is not sufficient for an application with many simultaneous users, however; it will likely cause duplicate values.

Figure 6 shows the procedures and queries used in addition to two panels. The first panel is the user data entry panel, and the second is an unseen interim panel called **MAX_VALUE**.

The application shown in Figure 6 was created to store customer feedback information. The first procedure, **MASTER_PROC**, begins the sequential numbering process. The first iteration of this process can be somewhat time consuming, as the system must load information into memory. It also requires a dummy row with a value of at least zero to be entered into the table. Once the table is populated, the dummy row can be deleted.

```
/*Procedure to run account branch Postinstall reports.*/
/*14W*/
"SET CURRENT (BRANCH='14W') "
'RUN QUERY POST_DEFECTS (REPORT=NO) '
'PRINT REPORT (FORM=POST_DEFECTS, WIDTH=135, PAGENO=NO, DATETIME=NO,
  LENGTH=52, FILE=R:\POSTINST\&BRANCH.ASC, CONFIRM=NO) '
/*28W*/
"SET CURRENT (BRANCH='28W') "
'RUN QUERY POST_DEFECTS (REPORT=NO) '
'PRINT REPORT (FORM=POST_DEFECTS, WIDTH=135, PAGENO=NO, DATETIME=NO,
  LENGTH=52, FILE=R:\POSTINST\&BRANCH.ASC, CONFIRM=NO) '
```

Figure 5: Linking queries

MASTER_PROC runs a query that selects the next sequential number. The value is saved in a temporary table and then the **MAX_VALUE** panel is run. The **MAX_VALUE** panel gets the value from the temporary table and places it into a global variable. There are two rules for this panel: the Panel Search Query rule runs the **GET_VALUE** query and the Procedure (Change) rule runs the **GET_VALUE** procedure.

Finally, the master procedure runs the actual data entry panel, known as **FEEDBACK_CARD**. This panel has a Procedure (Add) instance rule of **PUT_VALUE**, and the Add and Next action has been changed to run the procedure **ADD_PANEL**. **ADD_PANEL** is similar to the master procedure in that it processes a selection of the next sequential number so many rows can be entered without leaving the data entry panel.

Queries, panels, and procedures provide tremendous flexibility in application development. The Database Manager upon which they are built is very robust, providing needed security and integrity. These applications can be used either in a stand-alone mode or by many users on a local area network.





```

Procedure: MASTER_PROC
/*Master procedure for creating a sequential number in the problem_number field*/
'RUN QUERY max_value (INTERACT=no)' /*Selects next number */
'SAVE DATA AS max_value_table (COMMENT="Temporary Table", CONFIRM=no,REPLACE=yes)'

/*The next panel gets the value from the temporary table and puts it into a global variable*/
'RUN PANEL max_value (mode=change)'

'RUN PANEL FEEDBACK_CARD (mode=add)' /*Runs actual data entry panel*/

Procedure: GET_VALUE
/*The MAXVALUE panel displays the single next sequential problem number and sets a global
/*variable that will be picked up by the proc run by the actual FEEDBACK panel.*/

/*maxvalue is the name define in Table Fields*/
'GET CURRENT (var1=maxvalue)'
'SET GLOBAL (var2=var1)'
'QUIT PANEL'

Procedure: PUT_VALUE
/*These functions take the global variable set from the max_value panel and makes it a
/*current variable when this value is called from the actual data entry panel.*/
'GET GLOBAL (var3=var2)'
'SET CURRENT (KEY=var3)'

Procedure: ADD_PANEL
/*This procedure completes the first add of an item and then re-executes the function
/*to select the next sequential number and re-display the add panel*/
'ADD'
'RUN QUERY max_value (INTERACT=no)'
'SAVE DATA AS max_value_table (COMMENT="Temporary Table", CONFIRM=no, REPLACE=yes)'
'RUN PANEL max_value (mode=change)'

Query: MAX_VALUE
SELECT MAX(problem_number)+1 FROM feedback_table

Query: GET_VALUE
SELECT * FROM max_value_table

```

Figure 6: Sequential number routines

Alan S. Eisen, IBM Federal Systems Co., 800 N. Frederick Ave., Gaithersburg, Md., 20879-3395, (301) 240-6848. Eisen began his career as a systems engineer in federal marketing in 1987 and became an advisory marketing support representative managing Federal Marketing's systems assurance program in 1990. He is currently an advisory analyst in the Gaithersburg quality assurance department. Eisen received a B.A. in economics from the University of Maryland in 1981 and an M.S. in information systems from American University in 1986. Before joining IBM, he was a consultant with Wang Laboratories.

REFERENCES

IBM Extended Services for OS/2 Structured Query Language (SQL) Reference (IBM Doc. S04G-1012)

IBM Operating System/2 Extended Edition Version 1.2 User's Guide, Volume 3: Database Manager (IBM Doc. 15F7292)

IBM Operating System/2 Extended Edition Version 1.2 Database Manager Programming Guide and Reference (IBM Doc. S01F-0269)



The Only Tool You'll Ever Need

... to create, modify, manage, browse and translate resources such as dialog boxes, bitmaps, icons, cursors and menus.

Increase the productivity of your development work by organizing resources into projects and seamlessly integrate all the resource editors.

Reduce the amount of time required to design the user interface of your application—use the state of the art visual editors to interactively design and modify dialogs, bitmaps, icons, cursors, menus, accelerators, string tables and all other resources—or just use any of the hundreds of professionally designed resources included.

Capture images from anywhere on the screen directly into the image editor. Create bitmaps, icons and cursors from the captured image.

Customize the look and feel of your applications—directly extract, modify or replace any resource in any program file, EXE, DLL—no source code is needed.



Translate applications into foreign languages without

having access to the applications source code—

directly modify all messages, menus and dialogs into any national language.

Edit, copy and paste resources directly into and from any EXE, RES, DLL, RC program file. Eliminate the time consuming edit-compile-link cycle—increasing your productivity dramatically.

Migrate your applications to new 32-bit OS/2 2.0 and NT applications utilizing the transparent and automatic resource translation capabilities. Create your resources once on any platform and instantly use them on any other platform.

Project views and preview browsers allow you to efficiently manage your resources.



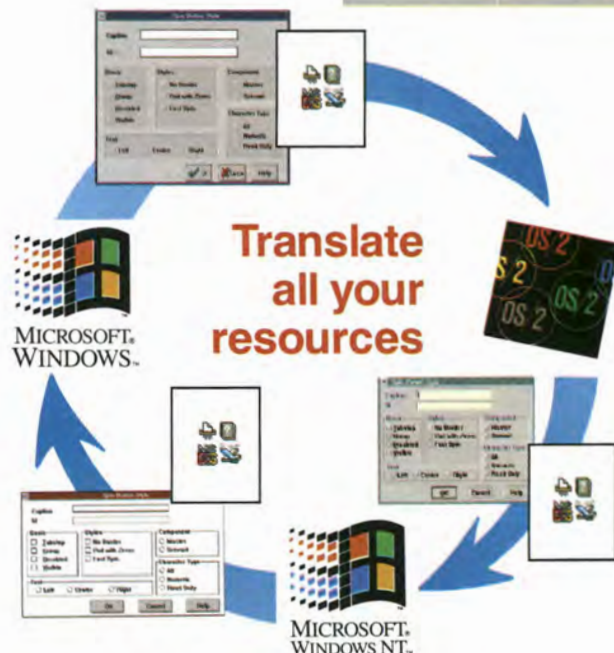
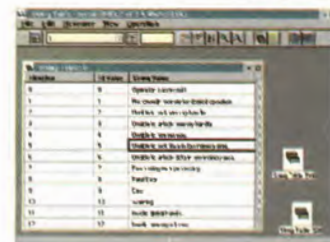
Quickly design comp dialogs using powerful design tools in Dialog Editor.

The Menu Editor.

Create bitmaps, icons, cursors using the advanced image editors.



Find, replace and translate text messages in string tables.



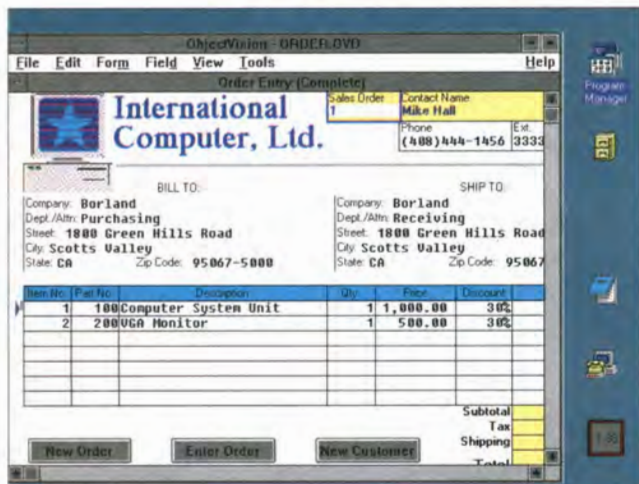
The Stirling Group
Making it easy to make™

172 OLD MILL DRIVE • SCHAMBURG, IL 60193 • USA **1-800-3 SHIELD**
1-800-374-4353

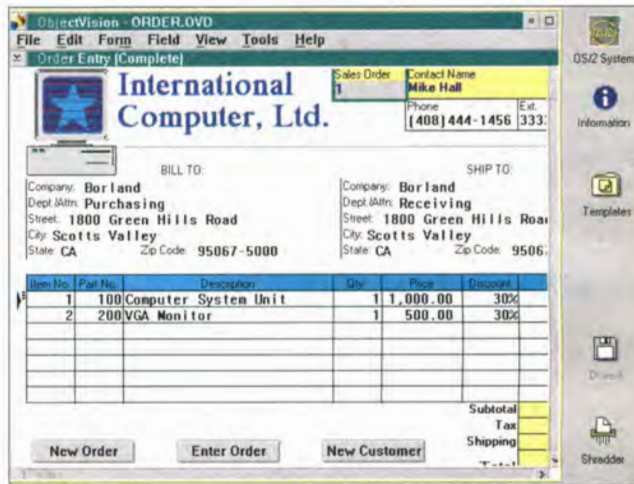
CALL 708-307-9197 • FAX 708-307-9340
CompuServe: GO STIRLING • MCI Mail: STIRLING

Q: What do Windows and OS/2 have in common?

New Version



ObjectVision for Windows



ObjectVision for OS/2

A: Your ObjectVision applications.

Borland's ObjectVision® brings Microsoft® Windows and OS/2® users together. Only ObjectVision allows you to write an application *once* for use under both Windows and OS/2. And access the same data, regardless of platform. That's because ObjectVision, the world's easiest tool for creating Windows applications, is now also available for OS/2.

Visual programming makes it easy



Using revolutionary visual programming techniques, ObjectVision makes it easy for the people who know the business issues to develop and maintain their own graphical software applications without programming.

Creating Windows or OS/2 applications is as easy as A-B-C. Simply:

- Draw your application interface using the convenient form tool.
- Apply your business rules visually.
- Connect your application to your existing database. Or create your own database.

In Windows and OS/2, ObjectVision makes it easy to create, modify, and run your own interactive applications. And the OS/2 version gives you easy access to REXX as well as DLLs.

Easy access to data

ObjectVision is the ideal front end to your data because it integrates information from different databases under one familiar, graphical user interface. Your ObjectVision applications can easily connect to information stored in Paradox®, dBASE®, Btrieve, Database Manager, and ASCII formats.

FREE runtime version

Because a runtime version is included *free*, it's easy to distribute your customized applications throughout your company. With ObjectVision, fully interoperable Windows and OS/2 applications are here today!

See your dealer today to get your own copy of ObjectVision for Windows or OS/2, OR call **1-800-331-0877, ext. 6622** to request your ObjectVision for Windows

FREE TRIAL DISK

In Canada, call **1-800-461-3327**



6362-0001-16



BORLAND
The Leader in Object-Oriented Programming

Copyright © 1992 Borland International, Inc. All rights reserved. ObjectVision, Paradox, and dBASE are registered trademarks of Borland International, Inc. BI 1552.1